# Practical 1 – Pre-processing of Text

In [1]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\anjali\anaconda3\lib\site-packages (3.6.1)
Requirement already satisfied: click in c:\users\anjali\anaconda3\lib\site-packages (from nltk) (7.1.2)
Requirement already satisfied: regex in c:\users\anjali\anaconda3\lib\site-packages (from nltk) (2021.4.4)
Requirement already satisfied: tqdm in c:\users\anjali\anaconda3\lib\site-packages (from nltk) (4.59.0)
Requirement already satisfied: joblib in c:\users\anjali\anaconda3\lib\site-packages (from nltk) (1.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import nltk
```

In [3]:
```
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[3]:
```
True
```

In [4]:
```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

In [5]:
```
text1
```

Out[5]:
```
<Text: Moby Dick by Herman Melville 1851>
```

In [6]:
```
text5
```

Out[6]:
```
<Text: Chat Corpus>
```

In [7]:
```
text1.concordance("monstrous") # to see word in context
```

```
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

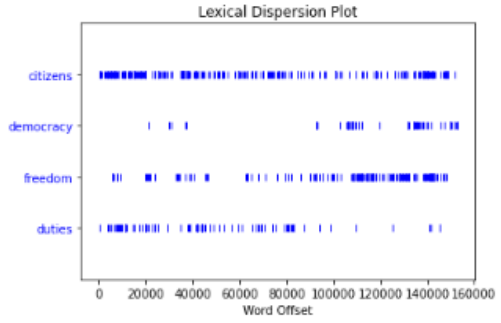
In [8]:
```
text1.similar("monstrous") # find similar words
```

```
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless
```

In [9]:
```
text2.common_contexts(["monstrous", "very"])
```

```
am_glad a_pretty a_lucky is_pretty be_glad
```

In [10]:
```
text4.dispersion_plot(["citizens", "democracy","freedom","duties"]) # position of words
```



In [11]:
```
text3.generate()
```

```
Building ngram index...
```

```
laid by her , and said unto Cain , Where art thou , and said , Go to ,
I will not do it for ten ' s sons ; we dreamed each man according to
their generatio the firstborn said unto Laban , Because I said , Nay ,
but Sarah shall her name be . , duke Elah , duke Shobal , and Akan .
and looked upon my affliction . Bashemath Ishmael ' s blood , but Isra
for as a prince hast thou found of all the cattle in the valley , and
the wo The
```

Out[11]:
```
"laid by her , and said unto Cain , Where art thou , and said , Go to ,\nI will not do it for ten ' s sons ; we dreamed eac
n according to\ntheir generatio the firstborn said unto Laban , Because I said , Nay ,\nbut Sarah shall her name be . , duk
ah , duke Shobal , and Akan .\nand looked upon my affliction . Bashemath Ishmael ' s blood , but Isra\nfor as a prince hast
u found of all the cattle in the valley , and\nthe wo The"
```

In [13]:
```
len(text3) # length vocabulary, punctuations, no spaces
```

Out[13]:
```
44764
```

In [14]:
```
# identify unique words and symbols
sorted(set(text3))
```

Out[14]:
```
['!',
 "'",
 '(',
 ')',
 ',',
 ',)',
 '.',
 '.)',
 ':',
 ';',
 ';)',
 '?',
 '?)',
 'A',
 'Abel',
 'Abelmizraim',
 'Abidah',
 'Abide',
 'Abimael',
 'Abimelech',
```

In [15]:
```
# count of unique tokens(word type)
len(set(text3))
```

Out[15]:
```
2789
```

In [16]:
```
len(text3)/len(set(text3)) #avg low --> used different words
```

Out[16]:
```
16.050197203298673
```

In [17]:
```
# count occurance of single word
text3.count("smote")
```

Out[17]:
```
5
```

In [18]:
```
# percentage
(text3.count("smote")/len(text3))*100
```

Out[18]:
```
0.01116968992940756
```

In [19]:
```
sent1=['call', 'me', 'Ismael', '.']
```

In [20]:
```
sent1.append("Some")
sent1
```

Out[20]:
```
['call', 'me', 'Ismael', '.', 'Some']
```

In [21]:
```
text4[173] # word at particular index
```

Out[21]:
```
'awaken'
```

In [22]:
```
text4.index("awaken") #index to word
```

Out[22]:
```
173
```

In [23]:
```
fdist1=FreqDist(text1) # freq dist of text
fdist1
```

Out[23]:
```
FreqDist({',': 18713, 'the': 13721, '.': 6862, 'of': 6536, 'and': 6024, 'a': 4569, 'to': 4542, ';': 4072, 'in': 3916, 'that': 2
982, ...})
```

In [24]:
```
vocab1=fdist1.keys()
```

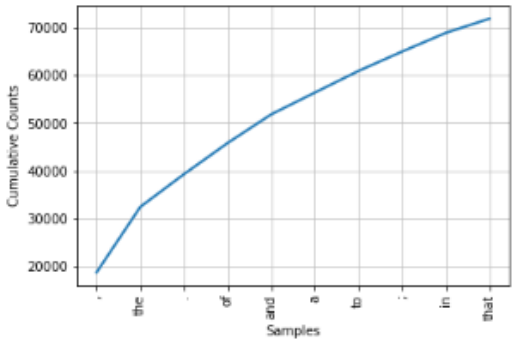
In [25]:
```
type(vocab1)
```

Out[25]:
```
dict_keys
```

In [26]:
```
list(vocab1)[:6] # 6 most used words
```

Out[26]:
```
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville']
```

In [27]:
```
#plot
fdist1.plot(10, cumulative=True)
```




Out[27]:
```
<AxesSubplot:xlabel='Samples', ylabel='Cumulative Counts'>
```

In [28]:
```
# words used only once
fdist1.hapaxes()
```

Out[28]:
```
['Herman',
 'Melville',
 ']',
 'ETYMOLOGY',
 'Late',
 'Consumptive',
 'School',
 'threadbare',
 'lexicons',
 'mockingly',
 'flags',
 'mortality',
 'signification',
 'HACKLUYT',
 'Sw',
 'HVAL',
 'roundness',
 'Dut',
 'Ger',
```

In [29]:
```
# fine grain of words

# find unique set of words
vocab2=set(text1)
```

In [30]:
```
long_words=[w for w in vocab2 if len(w)>15]
long_words
```

Out[30]:
```
['hermaphroditical',
 'CIRCUMNAVIGATION',
 'superstitiousness',
 'circumnavigation',
 'cannibalistically',
 'uncomfortableness',
 'uncompromisedness',
 'subterraneousness',
 'circumnavigations',
 'supernaturalness',
 'indiscriminately',
 'preternaturalness',
 'apprehensiveness',
 'irresistibleness',
 'uninterpenetratingly',
 'simultaneousness',
 'circumnavigating',
 'comprehensiveness',
 'responsibilities',
 'undiscriminating',
 'Physiognomically',
 'characteristically',
 'physiognomically',
 'indispensableness']
```

In [31]:
```
# long words (len>7) with freq>7
sorted([w for w in set(text1) if len(w)>7 and fdist1[w]>7])
```

```
 'compasses',
 'complete',
 'completely',
 'comrades',
 'concerned',
 'concerning',
 'conclude',
 'concluded',
 'concluding',
 'confidential',
 'connected',
 'connexion',
 'conscience',
 'conscious',
 'consider',
 'considerable',
 'considerably',
 'consideration',
 'considered',
```

In [32]:
```
# collocation(n-grams)
# all occurances of list of words with bi-grams
bgs=bigrams(['more', 'is', 'said', 'than', 'done'])
type(bgs)
```

Out[32]:
```
generator
```

In [33]:
```
for w in bgs: print(w)
```

```
('more', 'is')
('is', 'said')
('said', 'than')
('than', 'done')
```

In [34]:
```
# collocation--> frequent bigrams
text4.collocations()
```

```
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
```

In [35]:
```
# frequency based on length
fdist= FreqDist([len(w) for w in text1])
fdist
```

Out[35]:
```
FreqDist({3: 50223, 1: 47933, 4: 42345, 2: 38513, 5: 26597, 6: 17111, 7: 14399, 8: 9966, 9: 6428, 10: 3528, ...})
```

In [36]:
```
fdist.keys()
```

Out[36]:
```
dict_keys([1, 4, 2, 6, 8, 9, 11, 5, 7, 3, 10, 12, 13, 14, 16, 15, 17, 18, 20])
```

In [37]:
```
fdist.items()
```

Out[37]:
```
dict_items([(1, 47933), (4, 42345), (2, 38513), (6, 17111), (8, 9966), (9, 6428), (11, 1873), (5, 26597), (7, 14399), (3, 5022
3), (10, 3528), (12, 1053), (13, 567), (14, 177), (16, 22), (15, 70), (17, 12), (18, 1), (20, 1)])
```

In [38]:
```
fdist[9] # no of words with len 9
```

Out[38]:
```
6428
```

In [39]:
```
fdist.max() # max no of words with given length
```

Out[39]:
```
3
```

In [40]:
```
fdist.freq(3)*100  # % of words having len 3
```

Out[40]:
```
19.255882431878046
```

In [41]:
```
# freq of monstrous in text1
fdist1=FreqDist(text1)
fdist1.freq("monstrous")
```

Out[41]:
```
3.834076505162584e-05
```

In [42]:
```
len(text1)
```

Out[42]:
```
260819
```

In [43]:
```
# table of fdist
fdist1.tabulate()
```

In [43]:
```
# table of fdist
fdist1.tabulate()
```

```
    ,    the    .   of   and
a   to   ;      in  that     '
-   his  it     I   s        is
he  with was    as  "        all
for  this !     at  by       but
not  --   him   from be       on
so  whale one   you had      have
there  But  or  were now     which
?   me   like  The  their    are
they  an  some  then  my     when
upon  out into  man  ship    up
more  Ahab ."   no   them    ye
what  old  sea  would if     been
we  other over these will    its
And  down only such  head    though
boat  her  time any  who     long
very  It   than !"   about    said
,"  yet  still those before  great
has  two  t    seemed must   whale
```

In [44]:
```
#compare fdist
fdist1=FreqDist(text1)
fdist2=FreqDist(text2)
print(fdist1 < fdist2)
print(fdist1 > fdist2)
print(fdist1 == fdist2)
```

```
False
False
False
```

In [1]:
```
import nltk
from nltk.corpus import gutenberg
```

In [2]:
```
nltk.corpus.gutenberg.fileids() # files available in gutenberg corpus
```

Out[2]:
```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

In [3]:
```
emma=nltk.corpus.gutenberg.words('austen-emma.txt')
len(emma)
```

Out[3]:
```
192427
```

In [4]:
```
# find presence with context
emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt')) # for text within the file
emma.concordance("surprize")
```

```
Displaying 25 of 37 matches:
er father , was sometimes taken by surprize at his being still able to pity `
hem do the other any good ." " You surprize me ! Emma must do Harriet good : a
Knightley actually looked red with surprize and displeasure , as he stood up ,
r . Elton , and found to his great surprize , that Mr . Elton was actually on
d aid ." Emma saw Mrs . Weston ' s surprize , and felt that it must be great ,
father was quite taken up with the surprize of so sudden a journey , and his f
y , in all the favouring warmth of surprize and conjecture . She was , moreove
he appeared , to have her share of surprize , introduction , and pleasure . Th
ir plans ; and it was an agreeable surprize to her , therefore , to perceive t
talking aunt had taken me quite by surprize , it must have been the death of m
f all the dialogue which ensued of surprize , and inquiry , and congratulation
 the present . They might chuse to surprize her ." Mrs . Cole had many to agre
the mode of it , the mystery , the surprize , is more like a young woman ' s s
 to her song took her agreeably by surprize -- a second , slightly but correct
" " Oh ! no -- there is nothing to surprize one at all .-- A pretty fortune ;
t to be considered . Emma ' s only surprize was that Jane Fairfax should accep
of your admiration may take you by surprize some day or other ." Mr . Knightle
ation for her will ever take me by surprize .-- I never had a thought of her i
 expected by the best judges , for surprize -- but there was great joy . Mr .
 sound of at first , without great surprize . " So unreasonably early !" she w
d Frank Churchill , with a look of surprize and displeasure .-- " That is easy
; and Emma could imagine with what surprize and mortification she must be retu
tled that Jane should go . Quite a surprize to me ! I had not the least idea !
 . It is impossible to express our surprize . He came to speak to his father o
g engaged !" Emma even jumped with surprize ;-- and , horror - struck , exclai
```

In [5]:
```
for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
```

In [6]:
```
#vocab
set(emma)
```

Out[6]:
```
{'white',
 'chances',
 'corn',
```

In [7]:
```
#sentences
mac_sent=gutenberg.sents('shakespeare-macbeth.txt')
mac_sent
```

Out[7]:
```
[['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']'], ['Actus', 'Primus', '.'], ...]
```

In [8]:
```
from nltk.corpus import nps_chat
```

In [9]:
```
nps_chat.fileids() # files in nps_chat
```

Out[9]:
```
['10-19-20s_706posts.xml',
 '10-19-30s_705posts.xml',
 '10-19-40s_686posts.xml',
 '10-19-adults_706posts.xml',
 '10-24-40s_706posts.xml',
 '10-26-teens_706posts.xml',
 '11-06-adults_706posts.xml',
 '11-08-20s_705posts.xml',
 '11-08-40s_706posts.xml',
 '11-08-adults_705posts.xml',
 '11-08-teens_706posts.xml',
 '11-09-20s_706posts.xml',
 '11-09-40s_706posts.xml',
 '11-09-adults_706posts.xml',
 '11-09-teens_706posts.xml']
```

In [10]:
```
chatroom=nps_chat.posts('10-19-20s_706posts.xml')
chatroom[123] # specific post
```

Out[10]:
```
['i',
 'do',
 "n't",
 'want',
 'hot',
 'pics',
 'of',
 'a',
 'female',
 ',',
 'I',
 'can',
 'look',
 'in',
 'a',
 'mirror',
 '.']
```

In [11]:
```
from nltk.corpus import brown
```

In [12]:
```
brown.categories()
```

Out[12]:
```
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

In [13]:
```
# words from particular category
brown.words(categories='news')
```

Out[13]:
```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

In [14]:
```
# words from fileids
brown.fileids()
```

Out[14]:
```
['ca01',
 'ca02',
 'ca03',
 'ca04',
 'ca05',
 'ca06',
 'ca07'
```

In [15]:
```
brown.words(fileids='ca18')
```

Out[15]:
```
['``', 'A', 'Night', 'in', 'New', 'Orleans', "''", ...]
```

In [16]:
```
# sentences from any category
brown.sents(categories='news')
```

Out[16]:
```
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary',
'election', 'produced', '``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place', '.'], ['The', 'jury',
'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', ',', 'which', 'had', 'ove
r-all', 'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of',
'Atlanta', "''", 'for', 'the', 'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.'], ...]
```

In [17]:
```
brown.sents(categories='hobbies')
```

Out[17]:
```
[['Too', 'often', 'a', 'beginning', 'bodybuilder', 'has', 'to', 'do', 'his', 'training', 'secretly', 'either', 'because', 'hi
s', 'parents', "don't", 'want', 'sonny-boy', 'to', '``', 'lift', 'all', 'those', 'old', 'barbell', 'things', "''", 'because',
'``', "you'll", 'stunt', 'your', 'growth', "''", 'or', 'because', 'childish', 'taunts', 'from', 'his', 'schoolmates', ',', 'lik
e', '``', 'Hey', 'lookit', 'Mr.', 'America', ';', ';'], ['whaddya', 'gonna', 'do', 'with', 'all', 'those', 'muscles', '(', 'o
f', 'which', 'he', 'has', 'none', 'at', 'the', 'time', ')', "''", '?', '?'], ...]
```

In [18]:
```
brown.sents(categories='religion')
```

Out[18]:
```
[['As', 'a', 'result', ',', 'although', 'we', 'still', 'make', 'use', 'of', 'this', 'distinction', ',', 'there', 'is', 'much',
'confusion', 'as', 'to', 'the', 'meaning', 'of', 'the', 'basic', 'terms', 'employed', '.'], ['Just', 'what', 'is', 'meant', 'b
y', '``', 'spirit', "''", 'and', 'by', '``', 'matter', "''", '?', '?'], ...]
```

In [19]:
```
brown.sents(categories=['adventure', 'news', 'fiction'])
```

Out[19]:
```
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary',
'election', 'produced', '``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place', '.'], ['The', 'jury',
'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', ',', 'which', 'had', 'ove
r-all', 'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of',
'Atlanta', "''", 'for', 'the', 'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.'], ...]
```

In [20]:
```
# frequency dist for modal verbs
news_text=brown.words(categories='news')
fdist = nltk.FreqDist([w.lower() for w in news_text])
modals= ['can', 'could', 'might', 'must', 'would', 'will']
for m in modals:
    print (m +':', fdist[m])
```

```
can: 94
could: 87
might: 38
must: 53
would: 246
will: 389
```

In [21]:
```
# conditional fdist
#(genre, word)= condition, event
cfd=nltk.ConditionalFreqDist((genre, word)
                             for genre in brown.categories()
                             for word in brown.words(categories=genre))
```

In [22]:
```
cfd
```

Out[22]:
```
<ConditionalFreqDist with 15 conditions>
```

In [23]:
```
genres=['news', 'fiction', 'hobbies']
modals=['can', 'could', 'might', 'must', 'will']
cfd.tabulate(conditions=genres, samples=modals)
```

```
          can could might  must  will
   news    93    86    38    50   389
fiction    37   166    44    55    52
hobbies   268    58    22    83   264
```

In [24]:
```
from nltk.corpus import reuters
reuters.fileids() # train and test sets
```

```
 'test/15061',
 'test/15062',
 'test/15063',
 'test/15065',
 'test/15067',
 'test/15069',
 'test/15070',
 'test/15074',
 'test/15077',
 'test/15078',
 'test/15079',
 'test/15082',
```

In [25]:
```
reuters.categories()
```

Out[25]:
```
['acq',
 'alum',
 'barley',
 'bop',
 'carcass',
 'castor-oil',
 'cocoa',
 'coconut',
 'coconut-oil',
 'coffee',
 'copper',
```

In [26]:
```
reuters.categories('training/9865')
```

Out[26]:
```
['barley', 'corn', 'grain', 'wheat']
```

In [27]:
```
reuters.categories(['training/9865', 'training/9880'])
```

Out[27]:
```
['barley', 'corn', 'grain', 'money-fx', 'wheat']
```

In [28]:
```
reuters.fileids('barley')
```

```
 'training/2232',
 'training/3132',
 'training/3324',
 'training/395',
 'training/4280',
 'training/4296',
 'training/5',
 'training/501',
 'training/5467',
 'training/5610',
 'training/5640',
 'training/6626',
 'training/7205',
 'training/7579',
 'training/8213',
 'training/8257',
 'training/8759',
 'training/9865',
 'training/9958']
```

In [29]:
```
reuters.fileids(['barely', 'corn'])
```

Out[29]:
```
['test/14832',
 'test/14858',
 'test/15033',
 'test/15043',
 'test/15106',
 'test/15287',
 'test/15341',
 'test/15618',
 'test/15648',
 'test/15676',
 'test/15686',
 'test/15720',
 'test/15845',
 'test/15856',
 'test/15860',
 'test/15863',
 'test/15871',
 'test/15875',
 'test/15877',
```

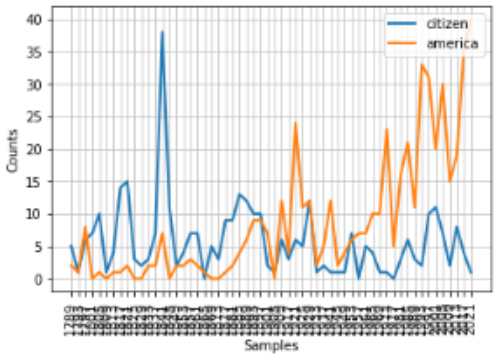
In [30]:
```
from nltk.corpus import inaugural
```

In [31]:
```
inaugural.fileids()
```

Out[31]:
```
['1789-Washington.txt',
 '1793-Washington.txt',
 '1797-Adams.txt',
 '1801-Jefferson.txt',
 '1805-Jefferson.txt',
 '1809-Madison.txt',
 '1813-Madison.txt',
 '1817-Monroe.txt',
 '1821-Monroe.txt',
 '1825-Adams.txt',
 '1829-Jackson.txt',
 '1833-Jackson.txt',
 '1837-VanBuren.txt',
```

In [32]:
```
cfd = nltk.ConditionalFreqDist(
    (target, fileid[:4]) # first 4 characters
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'citizen']
    if w.lower().startswith(target))
```

In [33]:
```
cfd.plot()
```




Out[33]: <AxesSubplot:xlabel='Samples', ylabel='Counts'>

In [34]:
```
nltk.corpus.cess_esp.words()
```

Out[34]: ['El', 'grupo', 'estatal', 'Electricité_de_France', ...]

In [35]:
```
nltk.corpus.indian.words('hindi.pos')
```

Out[35]: ['पूर्ण', 'प्रतिबंध', 'हटाओ', ':', 'इराक', 'संयुक्त', ...]

## Own corpus

In [36]:
```
from nltk.corpus import PlaintextCorpusReader
```

In [37]:
```
corpus_root = r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\OC'
```

In [38]:
```
wordlist=PlaintextCorpusReader(corpus_root, '.*')
```

In [39]:
```
wordlist.fileids()
```

Out[39]: ['c1.txt', 'c2.txt', 'c3.txt']

In [40]:
```
wordlist.words('c1.txt')
```

Out[40]: ['hii', 'hello', 'good', 'vibes', 'happy', 'people']

### Spell check

In [45]:
```
def unusual_words(text):
    text_vocab = set(w.lower() for w in text if w.isalpha())
    english_vocab = set(w.lower() for w in nltk.corpus.words.words())
    unusual = text_vocab.difference(english_vocab)
    return sorted(unusual)
```

In [46]:
```
unusual_words(nltk.corpus.gutenberg.words('austen-sense.txt'))
```

Out[46]:
```
['abbeyland',
 'abhorred',
 'abilities',
 'abounded',
 'abridgement',
 'abused',
 'abuses',
 'accents',
 'accepting',
 'accommodations',
 'accompanied',
 'accounted',
 'accounts',
 'accustomary',
 'aches',
```

```
In [47]: unusual_words(nltk.corpus.nps_chat.words('10-26-teens_706posts.xml'))
```

```
Out[47]: ['adds',
          'adduser',
          'ahhh',
          'alot',
          'anymore',
          'asking',
          'asl',
          'awesomee',
          'awwww',
          'aynawy',
          'beleive',
          'bi',
          'boinked',
          'bored',
          'brb',
          'bro',
          'bugs',
          'buying',
          'bwhaha',
```

### Puzzle Conditions:

book page 61

1. words of len>=4
2. no repition of words
3. R is in centre. Hence compulsory
4. no plurals/foreign words

**For words:**

```
In [48]: pl=nltk.FreqDist('egivrvonl')
```

```
In [49]: obligatory='r'
```

```
In [50]: wordlist=nltk.corpus.words.words()
```

```
In [51]: [w for w in wordlist if len(w) >= 4
                             and obligatory in w
                             and nltk.FreqDist(w) <= pl]
```

```
Out[51]: ['enrol',
          'ergon',
          'genro',
          'girl',
          'girn',
          'giro',
          'giver',
          'glor',
          'glore',
          'glover',
          'goer',
          'goner',
          'gore',
          'gorlin',
          'govern',
          'grein',
          'grin',
          'groin',
          'grove',
```

**For names:**

```
In [52]: names=nltk.corpus.names
```

```
In [53]: names.fileids()
```

```
Out[53]: ['female.txt', 'male.txt']
```

```
In [55]: male_names=names.words('male.txt')
         female_names=names.words('female.txt')
```

```
In [57]: common=[w for w in male_names if w in female_names]
         common
```

```
Out[57]: ['Abbey',
          'Abbie',
          'Abby',
          'Addie',
          'Adrian',
          'Adrien',
          'Ajay',
          'Alex',
          'Alexis',
          'Alfie',
          'Ali',
          'Alix',
          'Allie',
          'Allyn',
          'Andie',
          'Andrea',
          'Andy',
          'Angel',
          'Angie',
```

```
In [59]: len(common)
```

```
Out[59]: 365
```

## Pronouncing dictionary

In [60]:
```
entries=nltk.corpus.cmudict.entries()
```

In [61]:
```
len(entries)
```

Out[61]:
```
133737
```

In [63]:
```
for entry in entries[42371: 42382]:
    print(entry)
```

```
('fir', ['F', 'ER1'])
('fire', ['F', 'AY1', 'ER0'])
('fire', ['F', 'AY1', 'R'])
('firearm', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M'])
('firearm', ['F', 'AY1', 'R', 'AA2', 'R', 'M'])
('firearms', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M', 'Z'])
('firearms', ['F', 'AY1', 'R', 'AA2', 'R', 'M', 'Z'])
('fireball', ['F', 'AY1', 'ER0', 'B', 'AO2', 'L'])
('fireball', ['F', 'AY1', 'R', 'B', 'AO2', 'L'])
('fireballs', ['F', 'AY1', 'ER0', 'B', 'AO2', 'L', 'Z'])
('fireballs', ['F', 'AY1', 'R', 'B', 'AO2', 'L', 'Z'])
```

In [68]:
```
# sound of words starting with p and ending with T, having 3 sounds
for word, pron in entries:
    if len(pron)==3:
        ph1, ph2, ph3 =pron
        if ph1=='P' and ph3=='T':
            print (word, ph2)
```

```
pait EY1
pat AE1
pate EY1
patt AE1
peart ER1
peat IY1
peet IY1
peete IY1
pert ER1
pet EH1
pete IY1
pett EH1
piet IY1
piette IY1
pit IH1
pitt IH1
pot AA1
pote OW1
pott AA1
pout AW1
puett UW1
purt ER1
put UH1
putt AH1
```

In [71]:
```
# pronunciation of words ending with nics, nixs, niks
syllable = ['N', 'IH0', 'K', 'S']
n=[word for word, pron in entries if pron[-4:] == syllable] # last 4 characters must be in syllabe
n
```

Out[71]:
```
["atlantic's",
 'audiotronics',
 'avionics',
 'beatniks',
 'calisthenics',
 'centronics',
 'chamonix',
 'chetniks',
 "clinic's",
 'clinics',
```

In [72]:
```
len(n)
```

Out[72]:
```
83
```

In [75]:
```
# words ending with "mn"
[w for w,pron in entries if pron[-1]=='M' and w[-1]=='n']
# n--> last char in word
# M--> last char in pron
```

Out[75]:
```
['autumn', 'column', 'condemn', 'damn', 'goddamn', 'hymn', 'solemn']
```

In [80]:
```
# find all the p words consisting of three sounds
p3=[(pron[0]+'-'+pron[2], word)
for (word, pron) in entries if pron[0]=='P' and len(pron)==3]

cfd=nltk.ConditionalFreqDist(p3)

for temp in cfd.conditions():
    if len(cfd[temp]) >10:
        words=cfd[temp].keys()
        wordlist=''.join(words)
        print(temp, wordlist[:70])
```

```
P-P paappaapepappapepapppauppeeppeppippipepipppooppoppopepopppoppepup
P-R paarpairparpareparrpearpeerpierpoorpooreporporeporrpour
P-K pacpackpaekpaikpakpakepaquepeakpeakepechpeckpeekpercperkpicpickpikpike
P-S pacepasspastspeacepearsepeasepercepersperspepescepiecepisspospospostsp
P-L pahlpailpaillepalpalepallpaulpaulepaullpealpealepearlpearlepeelpeelepe
P-N paignpainpainepanpanepawnpaynepeinepenpenhpennpinpinepinnponpoonpunpyn
P-Z paispaizpao'spaspausepawspayspazpeaspeasepei'sperzpezpiespoe'spoisepos
P-T paitpatpatepattpeartpeatpeetpeetepertpetpetepettpietpiettepitpittpotpo
P-CH patchpautschpeachperchpetschpetschepichepiechpietschpitchpitschpoachpo
P-UW1 perupeughpewplewplueprewpruprueprughpshewpugh
```

In [1]:
```
import nltk
from nltk.corpus import *
from nltk import *
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

### text file

In [5]:
```
import urllib
```

In [11]:
```
from urllib import request
url = "https://www.gutenberg.org/files/2554/2554-0.txt"
```

In [17]:
```
raw=request.urlopen(url).read()
```

In [18]:
```
len(raw)
```

Out[18]:
```
1201520
```

In [19]:
```
raw[:75]
```

Out[19]:
```
b'\xef\xbb\xbfThe Project Gutenberg eBook of Crime and Punishment, by Fyodor Dostoevsk'
```

In [21]:
```
raw=raw.decode('utf-8')
```

In [22]:
```
tokens=nltk.word_tokenize(raw)
len(tokens)
```

Out[22]:
```
257058
```

In [23]:
```
tokens[:10]
```

Out[23]:
```
['\ufeffThe',
 'Project',
 'Gutenberg',
 'eBook',
 'of',
 'Crime',
 'and',
 'Punishment',
 ',',
 'by']
```

In [24]:
```
text=nltk.Text(tokens)
type(text)
```

Out[24]:
```
nltk.text.Text
```

In [25]:
```
text[1020:1060]
```

Out[25]:
```
['than',
 'we',
 'have',
 'his',
 'insight',
 'impresses',
 'us',
 'as',
 'wisdom',
 '...',
 'that',
 'wisdom',
 'of',
```

In [26]:
```
raw.find('PART I')
```

Out[26]:
```
5575
```

In [27]:
```
raw[5575]
```

Out[27]:
```
'P'
```

In [28]:
```
raw.rfind('end of project')  # 1st occurance of search from end of file
```

Out[28]:
```
-1
```

### Html extension

In [32]:
```
url='https://www.tutorialspoint.com/python/python_basic_syntax.htm'
html=request.urlopen(url).read()
```

In [33]:
```
html[:60]
```

Out[33]:
```
b'<!DOCTYPE html>\r\n<html lang="en-US">\r\n<head>\r\n<title>Python '
```

In [ ]:
```
'''from bs4 import BeautifulSoup
raw=nltk.get_text(html)
tokens=nltk.word_tokenize(raw)'''
```

### RSS feeds

In [34]:
```
pip install feedparser
```

```
Collecting feedparser
  Downloading feedparser-6.0.10-py3-none-any.whl (81 kB)
Collecting sgmllib3k
  Downloading sgmllib3k-1.0.0.tar.gz (5.8 kB)
Building wheels for collected packages: sgmllib3k
  Building wheel for sgmllib3k (setup.py): started
  Building wheel for sgmllib3k (setup.py): finished with status 'done'
  Created wheel for sgmllib3k: filename=sgmllib3k-1.0.0-py3-none-any.whl size=6065 sha256=4a9bc24235ae91fd584b959b5ddc9ef1505e4
c238db13bb8aca866f577b788a2
  Stored in directory: c:\users\anjali\appdata\local\pip\cache\wheels\83\63\2f\117884c3b19d46b64d3d61690333aa80c88dc14050e269c5
46
Successfully built sgmllib3k
Installing collected packages: sgmllib3k, feedparser
Successfully installed feedparser-6.0.10 sgmllib3k-1.0.0
Note: you may need to restart the kernel to use updated packages.
```

In [35]:
```
import feedparser
llog=feedparser.parse("http://languagelog.ldc.upenn.edu/nll/?feed=atom")
```

In [36]:
```
llog['feed']['title']
```

Out[36]:
```
'Language Log'
```

In [41]:
```
len(llog.entries)
```

Out[41]:
```
13
```

In [40]:
```
post=llog.entries[2]
post.title
```

Out[40]:
```
'Summer linguistics'
```

In [42]:
```
content=post.content[0].value
content[:70]
# Q. get clear text ???
```

Out[42]:
```
'<p>From Barbara Phillips Long:</p>\n<p style="padding-left: 40px;">In t'
```

In [54]:
```
f=open(r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\P4_raw.txt')
```

In [55]:
```
raw=f.read()
raw
```

Out[55]:
```
'hello world.\nweather is pleasant.\ngood vibes only'
```

In [51]:
```
# one line at a time
f=open(r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\P4_raw.txt','rU')
for line in f:
    print(line.strip())
```

```
hello world.
weather is pleasant.
good vibes only
```

```
<ipython-input-51-33111d12d78f>:2: DeprecationWarning: 'U' mode is deprecated
  f=open(r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\P4_raw.txt','rU')
```

In [58]:
```
s=input("Enter some text: ")
print("You typed", len(nltk.word_tokenize(s)),"words.")
```

```
Enter some text: hello world
You typed 2 words.
```

### textonyms

In [60]:
```
import re
wordlist=[w for w in nltk.corpus.words.words('en') if w.islower()]
[w for w in wordlist if re.search('^[ghi][mno][jkl][def]$',w)]
```

Out[60]:
```
['gold', 'golf', 'hold', 'hole']
```

In [62]:
```
# meta chars
# find words ending with ed from worldlist
x=[w for w in wordlist if re.search('ed$', w)]
x
```

Out[62]:
```
['abaissed',
 'abandoned',
 'abased',
 'abashed',
 'abatised',
 'abed',
 'aborted',
 'abridged',
 'abscessed',
 'absconded',
 'absorbed',
 'abstracted',
 'abstricted',
 'accelerated',
 'accepted',
 'accidented',
 'accoladed',
 'accolated',
 'accomplished',
```

In [63]:
```
len(x)
```

Out[63]:
```
9148
```

In [64]:
```
# search for pattern ..j..t..
y=[w for w in wordlist if re.search('^..j..t..$',w)]
y
```

Out[64]:
```
['abjectly',
 'adjuster',
 'dejected',
 'dejectly',
 'injector',
 'majestic',
 'objectee',
 'objector',
 'rejecter',
 'rejector',
 'unjilted',
```

In [65]:
```
len(y)
```

Out[65]:
```
13
```

In [66]:
```
chat_words=sorted(set(w for w in nltk.corpus.nps_chat.words()))
[w for w in chat_words if re.search('^m+i+n+e+$',w)]
```

Out[66]:
```
['miiiiiiiiiiiiinnnnnnnnnnneeeeeeeeee',
 'miiiiiinnnnnnnnnneeeeeeee',
 'mine',
 'mmmmmmmiiiiiiiiinnnnnnnnneeeeeeee']
```

In [67]:
```
# start a or h, can repeat any no of times
[w for w in chat_words if re.search('^[ah]+$',w)]
```

Out[67]:
```
['a',
 'aaaaaaaaaaaaaaaa',
 'aaahhhh',
 'ah',
 'ahah',
 'ahahah',
 'ahh',
 'ahhahahaha',
 'ahhh',
 'ahhhh',
 'ahhhhhh',
 'ahhhhhhhhhhhhh',
 'h',
 'ha',
 'haaa',
 'hah',
 'haha',
 'hahaaa',
 'hahah',
 'hahaha',
 'hahahaa',
 'hahahah',
 'hahahaha',
 'hahahahaaa',
 'hahahahahaha',
 'hahahahahahaha',
 'hahahahahahahahahahahahahahahaha',
 'hahahhahah',
 'hahhahahaha']
```

In [69]:
```
# decimal
wsj=sorted(set(nltk.corpus.treebank.words()))
d=[w for w in wsj if re.search('^[0-9]+\.[0-9]+$',w)]
d
```

Out[69]:
```
 '169.9',
 '17.3',
 '17.4',
 '17.5',
 '17.95',
 '1738.1',
 '176.1',
 '18.3',
 '18.6',
 '18.95',
 '185.9',
 '188.84',
 '19.3',
 '19.50',
 '19.6',
 '19.94',
 '19.95',
 '191.9',
 '2.07',
 '2.1',
```

In [70]:
```
len(d)
```

Out[70]:
```
481
```

In [73]:
```
# only 4 digits
f=[w for w in wsj if re.search('^[0-9]{4}$',w)]
f
```

Out[73]:
```
['1614',
 '1637',
 '1787',
 '1901',
 '1903',
 '1917'
```

In [74]:
```
len(f)
```

Out[74]:
```
56
```

In [75]:
```
# number - char len 3,4,5
c=[w for w in wsj if re.search('^[0-9]+-[a-z]{3,5}$',w)]
c
```

Out[75]:
```
['10-day',
 '10-lap',
 '10-year',
 '100-share',
 '12-point',
 '12-year',
 '14-hour',
```

In [76]:
```
len(c)
```

Out[76]: 31

In [78]:
```
p=[w for w in wsj if re.search('^[a-z]+-[a-z]+$',w)]
p
```

```
 'crane-safety',
 'credit-rating',
 'cross-border',
 'crystal-lattice',
 'current-carrying',
 'custom-chip',
 'day-care',
 'dead-eyed',
 'decade-long',
 'detective-story',
 'direct-investment',
 'direct-mail',
 'disaster-assistance',
 'dollar-denominated',
 'double-digit',
 'drag-down',
 'drop-in',
 'drop-off',
 'durable-goods',
```

In [79]:
```
len(p)
```

Out[79]: 516

In [81]:
```
#end with ed or ing
d=[w for w in wsj if re.search('(ed|ing)$',w)]
d
```

Out[81]:
```
['62%-owned',
 'Absorbed',
 'According',
 'Adopting',
 'Advanced',
 'Advancing',
 'Alfred',
 'Allied',
 'Annualized',
 'Anything',
 'Arbitrage-related',
 'Arbitraging',
 'Asked',
 'Assuming',
 'Atlanta-based',
 'Baking',
 'Banking',
 'Beginning',
 'Beijing',
```

In [86]:
```
word='supercalifragilisticexpialidocious'
x=re.findall(r'[aeiou]',word)
len(x)
```

Out[86]: 16

In [89]:
```
# combination of vowels
fd=nltk.FreqDist(vs for word in wsj for vs in re.findall(r'[aeiou]{2,}',word))
fd
```

Out[89]: FreqDist({'io': 549, 'ea': 476, 'ie': 331, 'ou': 329, 'ai': 261, 'ia': 253, 'ee': 217, 'oo': 174, 'ua': 109, 'au': 106, ...})

In [91]:
```
len(fd)
```

Out[91]: 43

In [92]:
```
fd.items()
```

Out[92]:
```
dict_items([('ea', 476), ('oi', 65), ('ou', 329), ('io', 549), ('ee', 217), ('ie', 331), ('ui', 95), ('ua', 109), ('ai', 261),
('ue', 105), ('ia', 253), ('ei', 86), ('iai', 1), ('oo', 174), ('au', 106), ('eau', 10), ('oa', 59), ('oei', 1), ('oe', 15),
('eo', 39), ('uu', 1), ('eu', 18), ('iu', 14), ('aii', 1), ('aiia', 1), ('ae', 11), ('aa', 3), ('oui', 6), ('ieu', 3), ('ao',
6), ('iou', 27), ('uee', 4), ('eou', 5), ('aia', 1), ('uie', 3), ('iao', 1), ('eei', 2), ('uo', 8), ('uou', 5), ('eea', 1), ('u
eui', 1), ('ioa', 1), ('ooi', 1)])
```

print text without any vowel expect first and last position

In [93]:
```
regexp=r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]'
def compress(word):
    pieces=re.findall(regexp, word)
    return''.join(pieces)
```

In [94]:
```
eng_udhr=nltk.corpus.udhr.words('English-Latin1')
print(nltk.tokenwrap(compress(w) for w in eng_udhr[:75]))
```

```
Unvrsl Dclrtn of Hmn Rghts Prmble Whrs rcgntn of the inhrnt dgnty and
of the eql and inlnble rghts of all mmbrs of the hmn fmly is the fndtn
of frdm , jstce and pce in the wrld , Whrs dsrgrd and cntmpt fr hmn
rghts hve rsltd in brbrs acts whch hve outrgd the cnscnce of mnknd ,
and the advnt of a wrld in whch hmn bngs shll enjy frdm of spch and
```

## stem ---> remove suffixes

In [95]:
```
def stem(word):
    for suffix in ['ing', 'ly', 'ed', 'ious']:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word
```

In [96]:
```
w='moved'
stem(w)
```

Out[96]:
```
'mov'
```

In [97]:
```
w='going'
stem(w)
```

Out[97]:
```
'go'
```

In [98]:
```
# find words in between a and man
from nltk.corpus import gutenberg, nps_chat
moby=nltk.Text((gutenberg.words('melville-moby_dick.txt')))
moby.findall(r"<a>(<.*>) <man>")
```

```
monied; nervous; dangerous; white; white; white; pious; queer; good;
mature; white; Cape; great; wise; wise; butterless; white; fiendish;
pale; furious; better; certain; complete; dismasted; younger; brave;
brave; brave; brave
```

In [100]:
```
# find 3 words ending with bro
chat=nltk.Text((nps_chat.words()))
chat.findall(r"<.*> <.*> <bro>")
```

```
you rule bro; telling you bro; u twizted bro
```

In [101]:
```
# seq of 3 or more words starting with L
chat.findall(r"<l.*>{3,} ")
```

```
lol lol lol; lmao lol lol; lol lol lol; la la la la la; la la la; la
la la; lovely lol lol love; lol lol lol.; la la la; la la la
```

## hypernyms

In [104]:
```
# words and their hypernyms
from nltk.corpus import brown
hl=nltk.Text(brown.words(categories=['hobbies','learned']))
hl.findall(r"<\w*> <and> <other> <\w*s>")
```

```
speed and other activities; water and other liquids; tomb and other
landmarks; Statues and other monuments; pearls and other jewels;
charts and other items; roads and other features; figures and other
objects; military and other areas; demands and other factors;
abstracts and other compilations; iron and other metals
```

In [105]:
```
# certain x has certain y
hl.findall(r"<\w*> <having> <\w*s>")
```

```
is having his; not having serious; affairs having religious
```

In [106]:
```
raw
```

Out[106]:
```
'hello world.\nweather is pleasant.\ngood vibes only'
```

## Predefined stemmers

In [113]:
```
# tokens from raw
tokens=nltk.word_tokenize(raw)
porter=nltk.PorterStemmer()
lan=nltk.LancasterStemmer()
[porter.stem(t) for t in tokens]
```

Out[113]:
```
['hello',
 'world',
 '.',
 'weather',
 'is',
 'pleasant',
 '.',
 'good',
 'vibe',
 'onli']
```

In [114]:
```
[lan.stem(t) for t in tokens]
```

Out[114]:
```
['hello', 'world', '.', 'weath', 'is', 'pleas', '.', 'good', 'vib', 'on']
```

In [115]:
```
f=open(r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\P4_p.txt')
raw=f.read()
raw
```

Out[115]:
```
"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in dat
a structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as w
ell as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax e
mphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encour
ages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or bi
nary form without charge for all major platforms, and can be freely distributed."
```

In [116]:
```
tokens=nltk.word_tokenize(raw)
porter=nltk.PorterStemmer()
lan=nltk.LancasterStemmer()
[porter.stem(t) for t in tokens]
```

Out[116]:
```
['python',
 'is',
 'an',
 'interpret',
 ',',
 'object-ori',
 ',',
 'high-level',
 'program',
 'languag',
 'with',
 'dynam',
 'semant',
 '.',
 'it',
 'high-level',
 'built',
 'in',
 'data',
```

In [117]:
```
[lan.stem(t) for t in tokens]
```

```
 'program',
 'modul',
 'and',
 'cod',
 'reus',
 '.',
 'the',
 'python',
 'interpret',
```

## Lemmatizer

In [118]:
```
wnl=nltk.WordNetLemmatizer()
```

In [121]:
```
f=open(r'C:\Users\Anjali\Documents\MS CS sem 3 Natural Language Processing\P4_raw.txt')
raw=f.read()
raw
```

Out[121]:
```
'hello world.\nweather is pleasant.\ngood vibes only'
```

In [123]:
```
[wnl.lemmatize(t) for t in raw]
```

Out[123]:
```
['h',
 'e',
 'l',
 'l',
 'o',
 ' ',
```

### Segment sentences

In [128]:
```
sents_token=nltk.data.load('tokenizers/punkt/english.pickle')
text=nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents=sents_token.tokenize(text)
pprint(pprint(sents[171:181]))
```

```
['In the wild events which were to follow this girl had no\n'
 'part at all; he never saw her again until all his tale was over.',
 'And yet, in some indescribable way, she kept recurring like a\n'
 'motive in music through all his mad adventures afterwards, and the\n'
 'glory of her strange hair ran like a red thread through those dark\n'
 'and ill-drawn tapestries of the night.',
 'For what followed was so\nimprobable, that it might well have been a dream.',
 'When Syme went out into the starlit street, he found it for the\n'
 'moment empty.',
 'Then he realised (in some odd way) that the silence\n'
 'was rather a living silence than a dead one.',
 'Directly outside the\n'
 'door stood a street lamp, whose gleam gilded the leaves of the tree\n'
 'that bent out over the fence behind him.',
 'About a foot from the\n'
 'lamp-post stood a figure almost as rigid and motionless as the\n'
 'lamp-post itself.',
 'The tall hat and long frock coat were black; the\n'
 'face, in an abrupt shadow, was almost as dark.',
 'Only a fringe of\n'
 'fiery hair against the light, and also something aggressive in the\n'
 'attitude, proclaimed that it was the poet Gregory.',
 'He had something\n'
 'of the look of a masked bravo waiting sword in hand for his foe.']
None
```

## Tagging

In [1]:
```
import nltk
from nltk.corpus import *
from nltk import *
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

In [2]:
```
# using tagger to identity
text=nltk.word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

Out[2]:
```
[('And', 'CC'),
 ('now', 'RB'),
 ('for', 'IN'),
 ('something', 'NN'),
 ('completely', 'RB'),
 ('different', 'JJ')]
```

In [3]:
```
text=nltk.Text(word.lower() for word in nltk.corpus.brown.words())
text.similar('woman')
'''text.similar() method takes a word w, finds all contexts w1w w2,
then finds all words w that appear in the same context, i.e. w1ww2.'''
```

```
man time day year car moment world house family child country boy
state job place way war girl work word
```

Out[3]:
```
'text.similar() method takes a word w, finds all contexts w1w w2,\nthen finds all words w that appear in the same context, i.e.
w1ww2.'
```

In [4]:
```
# create a token and tag it
tagged_token=nltk.tag.str2tuple('fly/NN')
tagged_token
```

Out[4]:
```
('fly', 'NN')
```

In [5]:
```
print(tagged_token[1]) # tag of token
print(tagged_token[0]) # tag
```

```
NN
fly
```

In [6]:
```
# tag for multiple tokens
sent='''The/AT grand/JJ jury/NN commented/VBO on/IN a/AT number/NN '''
[nltk.tag.str2tuple(t) for t in sent.split()]
```

Out[6]:
```
[('The', 'AT'),
 ('grand', 'JJ'),
 ('jury', 'NN'),
 ('commented', 'VBO'),
 ('on', 'IN'),
 ('a', 'AT'),
 ('number', 'NN')]
```

In [7]:
```
# tagging of existing corpuses
nltk.corpus.brown.tagged_words()
```

Out[7]:
```
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

In [8]:
```
nltk.corpus.sinica_treebank.tagged_words()
```

Out[8]:
```
[('一', 'Neu'), ('友情', 'Nad'), ('嘉珍', 'Nba'), ...]
```

In [9]:
```
nltk.corpus.indian.tagged_words()
```

Out[9]:
```
[('মাহিষের', 'NN'), ('সন্তান', 'NN'), (':', 'SYM'), ...]
```

In [15]:
```
# freqDist w.r.t tags
from nltk.corpus import brown
brown_news_tagged=nltk.corpus.brown.tagged_words(categories='news', tagset='universal')
```

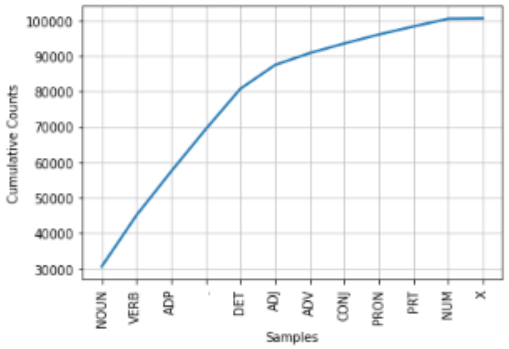
In [16]:
```
tag_fd=nltk.FreqDist(tag for (word,tag) in brown_news_tagged)
tag_fd
```

Out[16]:
```
FreqDist({'NOUN': 30654, 'VERB': 14399, 'ADP': 12355, '.': 11928, 'DET': 11389, 'ADJ': 6706, 'ADV': 3349, 'CONJ': 2717, 'PRON': 2535, 'PRT': 2264, ...})
```

In [17]:
```
tag_fd.keys()
```

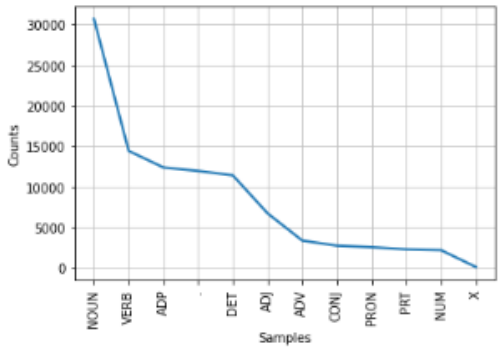
Out[17]:
```
dict_keys(['DET', 'NOUN', 'ADJ', 'VERB', 'ADP', '.', 'ADV', 'CONJ', 'PRT', 'PRON', 'NUM', 'X'])
```

In [18]:
```
tag_fd.plot(cumulative=True)
```




Out[18]:
```
<AxesSubplot:xlabel='Samples', ylabel='Cumulative Counts'>
```

In [19]:
```
tag_fd.plot()
```




Out[19]:
```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

In [34]:
```
# searching parts of speech before a noun
word_tag_pairs = list(nltk.bigrams(brown_news_tagged))
list(nltk.FreqDist(a[1] for (a, b) in word_tag_pairs if b[1] == 'NOUN'))
```

Out[34]:
```
['NOUN',
 'DET',
 'ADJ',
 'ADP',
 '.',
 'VERB',
 'CONJ',
 'NUM',
 'ADV',
```

In [29]:
```
word_tag_pairs
```

Out[29]:
```
[(('The', 'DET'), ('Fulton', 'NOUN')),
 (('Fulton', 'NOUN'), ('County', 'NOUN')),
 (('County', 'NOUN'), ('Grand', 'ADJ')),
 (('Grand', 'ADJ'), ('Jury', 'NOUN')),
 (('Jury', 'NOUN'), ('said', 'VERB')),
 (('said', 'VERB'), ('Friday', 'NOUN')),
 (('Friday', 'NOUN'), ('an', 'DET')),
 (('an', 'DET'), ('investigation', 'NOUN')),
 (('investigation', 'NOUN'), ('of', 'ADP')),
 (('of', 'ADP'), ("Atlanta's", 'NOUN')),
 (("Atlanta's", 'NOUN'), ('recent', 'ADJ')),
 (('recent', 'ADJ'), ('primary', 'NOUN')),
 (('primary', 'NOUN'), ('election', 'NOUN')),
 (('election', 'NOUN'), ('produced', 'VERB')),
 (('produced', 'VERB'), ('``', '.')),
 (('``', '.'), ('no', 'DET')),
 (('no', 'DET'), ('evidence', 'NOUN')),
 (('evidence', 'NOUN'), ("''", '.')),
 (("''", '.'), ('that', 'ADP')),
```

In [36]:
```
# for verb
list(nltk.FreqDist(a[1] for (a, b) in word_tag_pairs if b[1] == 'VERB'))
```

Out[36]:
```
['NOUN',
 'VERB',
 'PRON',
 'PRT',
 '.',
 'ADV',
 'DET',
 'CONJ',
 'ADP',
 'ADJ',
 'NUM',
 'X']
```

In [40]:
```
# most common verbs in text
wsj=nltk.corpus.treebank.tagged_words(tagset='universal')
wsj
```

Out[40]:
```
[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ...]
```

In [44]:
```
word_tag_fd=nltk.FreqDist(wsj)
[word + "/" + tag for (word, tag) in word_tag_fd if tag.startswith('V')]
```

```
'had/VERB',
'been/VERB',
'could/VERB',
"'s/VERB",
'can/VERB',
'do/VERB',
'say/VERB',
'make/VERB',
'may/VERB',
'did/VERB',
'rose/VERB',
'made/VERB',
'does/VERB',
'expected/VERB',
'buy/VERB',
'take/VERB',
'get/VERB',
'might/VERB',
'sell/VERB',
```

In [47]:
```
# word--> condition, tag--> event
cfd1=nltk.ConditionalFreqDist(wsj)
cfd1['yield'].keys()
```

Out[47]:
```
dict_keys(['NOUN', 'VERB'])
```

In [46]:
```
cfd1['cut'].keys()
```

Out[46]:
```
dict_keys(['VERB', 'NOUN'])
```

In [54]:
```
cfd2 = nltk.ConditionalFreqDist((tag, word) for (word, tag) in wsj)
cfd2['VERB'].keys()
```

```
g', 'cover', 'switch', 'phase', 'equals', 'counting', 'fabricate', 'Related', 'converting', 'speculate', 'positioned', 'fac
ing', 'buoyed', 'note', 'underpin', 'kept', 'plunging', 'locked', 'offset', 'resulting', 'reasons', 'driving', 'convinced',
'starts', 'erode', 'traced', 'waiting', 'contends', 'mollified', 'cites', 'valued', 'redeeming', 'rolled', 'point', 'lock',
'recede', 'drifted', 'measures', 'Estimated', 'keeps', 'laughing', 'air', 'fuming', 'sign', 'risk', 'losing', 'persuade',
'negotiate', 'tell', "'ll", 'flooded', 'drop', 'renew', 'pleased', 'adds', 'fawning', 'survive', 'thumbing', 'billed', 'Fou
nded', 'combines', 'happens', 'flush', 'identify', 'deem', 'alienated', 'chastised', 'pointing', 'wrote', 'practicing', 'fu
mes', 'printed', 'portrayed', 'advertise', 'question', 'turning', 'spend', 'relied', 'recycled', 'scared', 'replies', 'car
e', 'sleep', 'expanded', 'supply', 'handled', 'serviced', 'breaks', 'specializes', 'damaged', 'Developed', 'pitches', 'pres
sed', 'ward', 'sweetened', 'allocated', 'leaving', 'raises', 'top', 'criticized', 'characterizing', 'entrench', 'confuse',
'materialize', 'converted', 'place', 'mired', 'barred', 'responding', 'requiring', 'equip', 'represents', 'equipped', 'urgi
ng', 'classed', 'address', 'praised', 'noting', 'weighing', 'withstand', 'depressed', 'phasing', 'installed', 'installing',
'enclosed', 'transporting', 'joins', 'retires', 'prevailing', 'lay', 'reduced', 'succeeding', 'retiring', 'declared', 'sing
```

In [56]:
```
cfd2['NOUN'].keys()
```

Out[56]:
```
dict_keys(['Pierre', 'Vinken', 'years', 'board', 'director', 'Nov.', 'Mr.', 'chairman', 'Elsevier', 'N.V.', 'Dutch', 'grou
p', 'Rudolph', 'Agnew', 'Consolidated', 'Gold', 'Fields', 'PLC', 'conglomerate', 'form', 'asbestos', 'Kent', 'cigarette',
'filters', 'percentage', 'cancer', 'deaths', 'workers', 'researchers', 'fiber', 'crocidolite', 'lungs', 'exposures', 'sympt
oms', 'decades', 'Lorillard', 'Inc.', 'unit', 'Loews', 'Corp.', 'cigarettes', 'Micronite', 'findings', 'year', 'results',
'today', 'New', 'England', 'Journal', 'Medicine', 'forum', 'attention', 'problem', 'spokewoman', 'story', 'anyone', 'proper
ties', 'products', 'research', 'smokers', 'information', 'users', 'risk', 'James', 'A.', 'Talcott', 'Boston', 'Dana-Farbe
r', 'Cancer', 'Institute', 'Dr.', 'team', 'National', 'schools', 'Harvard', 'University', 'spokeswoman', 'amounts', 'pape
r', 'type', 'filter', 'company', 'men', 'substance', 'times', 'number', 'diseases', 'total', 'mesothelioma', 'lung', 'asbes
tosis', 'morbidity', 'rate', 'finding', 'West', 'Groton', 'Mass.', 'factory', 'countries', 'plant', 'Hollingsworth', 'Vos
```

In [57]:
```
cfd2['factory']
```

Out[57]:
```
FreqDist({})
```

In [62]:
```
#  most frequent nouns of each noun part-of-speech type
def findtags(tag_prefix, tagged_text):
    cfd=nltk.ConditionalFreqDist((tag, word) for (word, tag) in tagged_text if tag.startswith(tag_prefix))
    return dict((tag, cfd[tag].most_common(5)) for tag in cfd.conditions())
```

In [63]:
```
tagdict = findtags('NN', nltk.corpus.brown.tagged_words(categories='news'))
for tag in sorted(tagdict):
    print (tag, tagdict[tag])
```

```
NN [('year', 137), ('time', 97), ('state', 88), ('week', 85), ('man', 72)]
NN$ [("year's", 13), ("world's", 8), ("state's", 7), ("nation's", 6), ("city's", 6)]
NN$-HL [("Golf's", 1), ("Navy's", 1)]
NN$-TL [("President's", 11), ("Administration's", 3), ("Army's", 3), ("League's", 3), ("University's", 3)]
NN-HL [('sp.', 2), ('problem', 2), ('Question', 2), ('cut', 2), ('party', 2)]
NN-NC [('ova', 1), ('eva', 1), ('aya', 1)]
NN-TL [('President', 88), ('House', 68), ('State', 59), ('University', 42), ('City', 41)]
NN-TL-HL [('Fort', 2), ('Mayor', 1), ('Commissioner', 1), ('City', 1), ('Oak', 1)]
NNS [('years', 101), ('members', 69), ('people', 52), ('sales', 51), ('men', 46)]
NNS$ [("children's", 7), ("women's", 5), ("men's", 3), ("janitors'", 3), ("taxpayers'", 2)]
NNS$-HL [("Dealers'", 1), ("Idols'", 1)]
NNS$-TL [("Women's", 4), ("States'", 3), ("Giants'", 2), ("Princes'", 1), ("Bombers'", 1)]
NNS-HL [('Wards', 1), ('deputies', 1), ('bonds', 1), ('aspects', 1), ('Decisions', 1)]
NNS-TL [('States', 38), ('Nations', 11), ('Masters', 10), ('Communists', 9), ('Rules', 9)]
NNS-TL-HL [('Nations', 1)]
```

In [66]:
```
# how word is used in text
brown_learned_text=brown.words(categories='learned')
sorted(set(b for (a,b) in nltk.bigrams(brown_learned_text) if a == 'often'))
```

Out[66]:
```
[',',
 '.',
 'accomplished',
 'analytically',
 'appear',
 'apt',
 'associated',
 'assuming',
 'became',
 'become',
 'been',
 'began',
 'call',
 'called',
 'carefully',
 'chose',
 'classified',
 'colorful',
 'composed',
 'contain',
 'differed',
 'difficult',
 'encountered',
 'enough',
 'equate',
 'extremely',
 'found',
 'happens',
 'have',
 'ignored',
 'in',
 'involved',
 'more',
 'needed',
 'nightly',
 'observed',
 'of',
 'on',
 'out',
```

In [68]:
```
# find trigram pattern using part of speech
def process(sentence):
    for (w1,t1), (w2,t2), (w3,t3) in nltk.trigrams(sentence):
        if (t1.startswith('V') and t2 == 'TO' and t3.startswith('V')):
            print(w1, w2, w3)
for tagged_sent in brown.tagged_sents():
    process(tagged_sent)
```

```
combined to achieve
continue to place
serve to protect
wanted to wait
allowed to place
expected to become
expected to approve
expected to make
```

In [69]:
```
# diffrent parts of speeches for words in corpus
brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
data = nltk.ConditionalFreqDist((word.lower(), tag)
                                for (word, tag) in brown_news_tagged)
for word in sorted(data.conditions()):
     if len(data[word]) > 3:
        tags = [tag for (tag, _) in data[word].most_common()]
        print(word, ' '.join(tags))
```

```
best ADJ ADV VERB NOUN
close ADV ADJ VERB NOUN
open ADJ VERB NOUN ADV
present ADJ ADV NOUN VERB
that ADP DET PRON ADV
```

### Automatic tagging

In [70]:
```
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
```

In [73]:
```
# default tagger
tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
nltk.FreqDist(tags).max()
```

Out[73]:
```
'NN'
```

In [75]:
```
raw = 'I do not like green eggs and ham, I do not like them Sam I am!'
tokens = nltk.word_tokenize(raw)
default_tagger = nltk.DefaultTagger('NN') # only 1 tag for all
default_tagger.tag(tokens)
```

Out[75]:
```
[('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('green', 'NN'),
 ('eggs', 'NN'),
 ('and', 'NN'),
 ('ham', 'NN'),
 (',', 'NN'),
 ('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('them', 'NN'),
 ('Sam', 'NN'),
 ('I', 'NN'),
 ('am', 'NN'),
 ('!', 'NN')]
```

In [76]:
```
default_tagger.evaluate(brown_tagged_sents)
```

Out[76]:
```
0.13089484257215028
```

### Regular expression tagger

In [77]:
```
patterns = [
    (r'.*ing$', 'VBG'),               # gerunds
    (r'.*ed$', 'VBD'),                # simple past
    (r'.*es$', 'VBZ'),                # 3rd singular present
    (r'.*ould$', 'MD'),               # modals
    (r'.*\'s$', 'NN$'),               # possessive nouns
    (r'.*s$', 'NNS'),                 # plural nouns
    (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'),  # cardinal numbers
    (r'.*', 'NN') ]                    # nouns (default)
```

In [78]:
```
regexp_tagger = nltk.RegexpTagger(patterns)
regexp_tagger.tag(brown_sents[3])
```

Out[78]:
```
[('``', 'NN'),
 ('Only', 'NN'),
 ('a', 'NN'),
 ('relative', 'NN'),
 ('handful', 'NN'),
 ('of', 'NN'),
 ('such', 'NN'),
 ('reports', 'NNS'),
 ('was', 'NNS'),
 ('received', 'VBD'),
 ("''", 'NN'),
 (',', 'NN'),
 ('the', 'NN'),
 ('jury', 'NN'),
 ('said', 'NN'),
 (',', 'NN'),
 ('``', 'NN'),
 ('considering', 'VBG'),
 ('the', 'NN'),
 ('widespread', 'NN'),
 ('interest', 'NN'),
 ('in', 'NN'),
 ('the', 'NN'),
 ('election', 'NN'),
 (',', 'NN'),
 ('the', 'NN'),
 ('number', 'NN'),
 ('of', 'NN'),
 ('voters', 'NNS'),
 ('and', 'NN'),
 ('the', 'NN'),
 ('size', 'NN'),
 ('of', 'NN'),
 ('this', 'NNS'),
 ('city', 'NN'),
 ("''", 'NN'),
 ('.', 'NN')]
```

In [79]:
```
regexp_tagger.evaluate(brown_tagged_sents)
```

Out[79]:
```
0.20186168625812995
```

### Lookup tager

- A lot of high-frequency words do not have the NN tag

In [80]:
```
# for tagged words
fd = nltk.FreqDist(brown.words(categories='news'))
cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
most_freq_words = fd.most_common(100) # 100 most freq words
```

In [81]:
```
likely_tags = dict((word, cfd[word].max()) for (word, _) in most_freq_words)
```

In [82]:
```
baseline_tagger = nltk.UnigramTagger(model=likely_tags)
baseline_tagger.evaluate(brown_tagged_sents)
```

Out[82]:
```
0.45578495136941344
```

In [83]:
```
# for untagged input text
sent=brown.sents(categories='news')[3]
baseline_tagger.tag(sent)
```

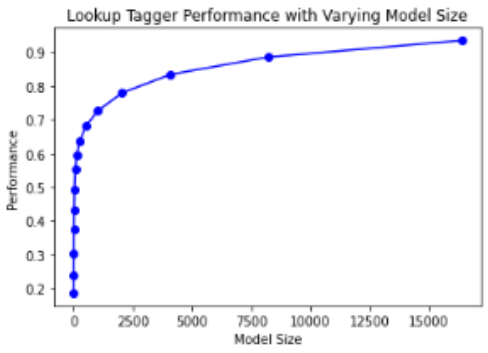
Out[83]:
```
[('``', '``'),
 ('Only', None),
 ('a', 'AT'),
 ('relative', None),
 ('handful', None),
 ('of', 'IN'),
 ('such', None),
 ('reports', None),
 ('was', 'BEDZ'),
 ('received', None),
 ("''", "''"),
 (',', ','),
 ('the', 'AT'),
 ('jury', None),
 ('said', 'VBD'),
 (',', ','),
 ('``', '``'),
 ('considering', None),
 ('the', 'AT'),
 ('widespread', None),
 ('interest', None),
 ('in', 'IN'),
 ('the', 'AT'),
 ('election', None),
 (',', ','),
 ('the', 'AT'),
 ('number', None),
 ('of', 'IN'),
 ('voters', None),
 ('and', 'CC'),
 ('the', 'AT'),
 ('size', None),
 ('of', 'IN'),
 ('this', 'DT'),
 ('city', None),
 ("''", "''"),
 ('.', '.')]
```

In [84]:
```
#None --> not among the 100 most frequent words
# assign NN using backoff
baseline_tagger = nltk.UnigramTagger(model=likely_tags, backoff=nltk.DefaultTagger('NN'))
```

In [86]:
```
def performance(cfd, wordlist):
    lt = dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model=lt, backoff=nltk.DefaultTagger('NN'))
    return baseline_tagger.evaluate(brown.tagged_sents(categories='news'))

def display():
    import pylab
    word_freqs = nltk.FreqDist(brown.words(categories='news')).most_common()
    words_by_freq = [w for (w, _) in word_freqs]
    cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
    sizes = 2 ** pylab.arange(15)
    perfs = [performance(cfd, words_by_freq[:size]) for size in sizes]
    pylab.plot(sizes, perfs, '-bo')
    pylab.title('Lookup Tagger Performance with Varying Model Size')
    pylab.xlabel('Model Size')
    pylab.ylabel('Performance')
    pylab.show()
```

In [87]:
```
display()
```



## Unigram tagging

In [88]:
```
from nltk.corpus import brown
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
unigram_tagger.tag(brown_sents[2007])
```

Out[88]:
```
[('Various', 'JJ'),
 ('of', 'IN'),
 ('the', 'AT'),
 ('apartments', 'NNS'),
 ('are', 'BER'),
 ('of', 'IN'),
 ('the', 'AT'),
 ('terrace', 'NN'),
 ('type', 'NN'),
 (',', ','),
 ('being', 'BEG'),
 ('on', 'IN'),
 ('the', 'AT'),
 ('ground', 'NN'),
 ('floor', 'NN'),
 ('so', 'QL'),
 ('that', 'CS'),
 ('entrance', 'NN'),
 ('is', 'BEZ'),
 ('direct', 'JJ'),
 ('.', '.')]
```

In [89]:
```
unigram_tagger.evaluate(brown_tagged_sents)
```

Out[89]:
```
0.9349006503968017
```

In [90]:
```
# separating trainig and test data
size=int(len(brown_tagged_sents)*0.9)
size
```

Out[90]:
```
4160
```

In [91]:
```
train_sents=brown_tagged_sents[:size]
test_sents=brown_tagged_sents[size:]
```

In [93]:
```
unigram_tagger=nltk.UnigramTagger(train_sents)
unigram_tagger.evaluate(test_sents)
```

Out[93]:
```
0.8121200039868434
```

## General N-gram tagger

In [94]:
```
bigram_tagger = nltk.BigramTagger(train_sents)
bigram_tagger.tag(brown_sents[2007])
```

Out[94]:
```
[('Various', 'JJ'),
 ('of', 'IN'),
 ('the', 'AT'),
 ('apartments', 'NNS'),
 ('are', 'BER'),
 ('of', 'IN'),
 ('the', 'AT'),
 ('terrace', 'NN'),
 ('type', 'NN'),
 (',', ','),
 ('being', 'BEG'),
 ('on', 'IN'),
 ('the', 'AT'),
 ('ground', 'NN'),
 ('floor', 'NN'),
 ('so', 'CS'),
 ('that', 'CS'),
 ('entrance', 'NN'),
 ('is', 'BEZ'),
 ('direct', 'JJ'),
 ('.', '.')]
```

In [95]:
```
unseen_sent = brown_sents[4203]
bigram_tagger.tag(unseen_sent)
```

Out[95]:
```
[('The', 'AT'),
 ('population', 'NN'),
 ('of', 'IN'),
 ('the', 'AT'),
 ('Congo', 'NP'),
 ('is', 'BEZ'),
 ('13.5', None),
 ('million', None),
 (',', None),
 ('divided', None),
 ('into', None),
 ('at', None),
 ('least', None),
 ('seven', None),
 ('major', None),
 ('``', None),
 ('culture', None),
 ('clusters', None),
 ("''", None),
 ('and', None),
 ('innumerable', None),
 ('tribes', None),
 ('speaking', None),
 ('400', None),
 ('separate', None),
 ('dialects', None),
 ('.', None)]
```

In [96]:
```
bigram_tagger.evaluate(test_sents)
```

Out[96]:
```
0.10206319146815508
```

### Combinig tagger

In [100]:
```
# using most likely used tags
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(train_sents, backoff=t0)
t2 = nltk.BigramTagger(train_sents, backoff=t1)
t2.evaluate(test_sents)
```

Out[100]:
```
0.8452108043456593
```

In [99]:
```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(train_sents, backoff=t0)
t2 = nltk.BigramTagger(train_sents, backoff=t1)
t3 = nltk.TrigramTagger(train_sents, backoff=t2)
t3.evaluate(test_sents)
```

Out[99]:
```
0.843317053722715
```

### Storing taggers

- store statistical learning and apply on other text

In [101]:
```
# save tagger t2 to a file t2.pkl
from pickle import dump
output = open('t2.pkl', 'wb')
dump(t2, output, -1)
output.close()
```

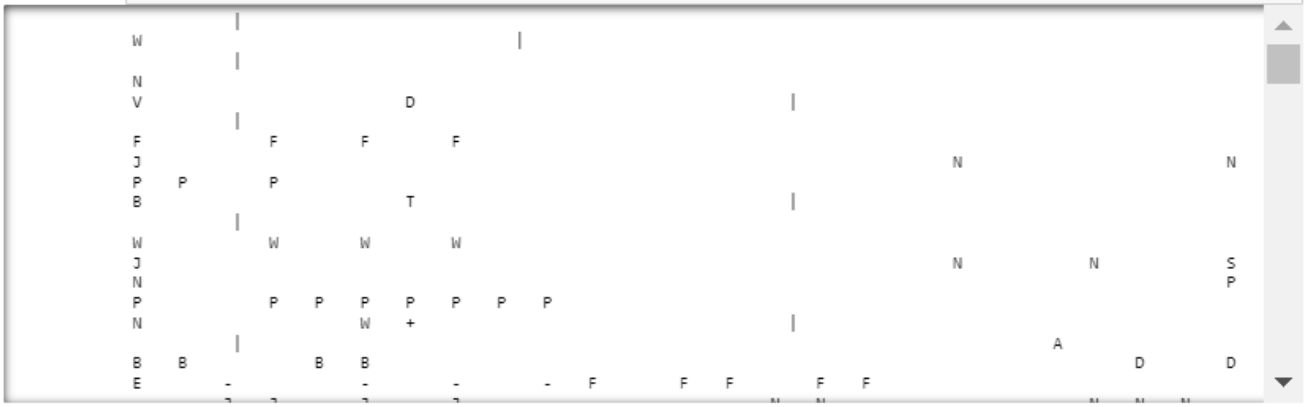
In [102]:
```
#  load saved tagger
from pickle import load
input = open('t2.pkl', 'rb')
tagger = load(input)
input.close()
```

In [103]:
```
text = """The board's action shows what free enterprise is up against in our complex maze of regulatory laws ."""
tokens = text.split()
tagger.tag(tokens)
```

Out[103]:
```
[('The', 'AT'),
 ("board's", 'NN$'),
 ('action', 'NN'),
 ('shows', 'NNS'),
 ('what', 'WDT'),
 ('free', 'JJ'),
 ('enterprise', 'NN'),
 ('is', 'BEZ'),
 ('up', 'RP'),
 ('against', 'IN'),
 ('in', 'IN'),
 ('our', 'PP$'),
 ('complex', 'JJ'),
 ('maze', 'NN'),
 ('of', 'IN'),
 ('regulatory', 'NN'),
 ('laws', 'NNS'),
 ('.', '.')]
```

### Confusion matrix

In [104]:
```
test_tags = [tag for sent in brown.sents(categories='editorial') for (word, tag) in t2.tag(sent)]
gold_tags = [tag for (word, tag) in brown.tagged_words(categories='editorial')]
print(nltk.ConfusionMatrix(gold_tags, test_tags))
```



## Practical 3 – Latent Semantic Analysis

```
! pip install ordered-set
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ordered-set
  Downloading ordered_set-4.1.0-py3-none-any.whl (7.6 kB)
Installing collected packages: ordered-set
Successfully installed ordered-set-4.1.0
```

```
import pandas as pd
import numpy as np
from collections import Counter
from ordered_set import OrderedSet
```

```
d1 = "The cow jumped over the moon"
d2 = "O'Leary's cow kicked the lamp"
d3 = "The kicked lamp started a fire"
d4 = "The cow on fire"
```

```
def term_doc(*docs):
    dd = []
    for doc in docs:
        dd.append(list(map(lambda x : x.lower(), doc.split(' '))))

    counts = dict()
    for idx, doc in enumerate(dd):
        counts[idx] = Counter(doc)

    mat =  pd.DataFrame(counts).replace(np.nan, 0)

    words = list(OrderedSet([word for doc in dd for word in doc]))

    return mat, words
```

```
mat, words = term_doc(d1, d2, d3, d4)
mat
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **the** | 2.0 | 1.0 | 1.0 | 1.0 |
| **cow** | 1.0 | 1.0 | 0.0 | 1.0 |
| **jumped** | 1.0 | 0.0 | 0.0 | 0.0 |
| **over** | 1.0 | 0.0 | 0.0 | 0.0 |
| **moon** | 1.0 | 0.0 | 0.0 | 0.0 |
| **o'leary's** | 0.0 | 1.0 | 0.0 | 0.0 |
| **kicked** | 0.0 | 1.0 | 1.0 | 0.0 |
| **lamp** | 0.0 | 1.0 | 1.0 | 0.0 |
| **started** | 0.0 | 0.0 | 1.0 | 0.0 |
| **a** | 0.0 | 0.0 | 1.0 | 0.0 |
| **fire** | 0.0 | 0.0 | 1.0 | 1.0 |
| **on** | 0.0 | 0.0 | 0.0 | 1.0 |

```
u, s, vt = np.linalg.svd(mat, full_matrices=False)
```

```
u, s, vt
```

```
(array([[-0.70701377, -0.19127925, -0.05030678, -0.11782658],
        [-0.40921666, -0.21126922,  0.07205224,  0.48568274],
        [-0.17254364, -0.29470143,  0.04273557, -0.27369126],
        [-0.17254364, -0.29470143,  0.04273557, -0.27369126],
        [-0.17254364, -0.29470143,  0.04273557, -0.27369126],
        [-0.12934189,  0.11257205,  0.4615164 ,  0.31939271],
        [-0.25459535,  0.42726345,  0.29642181, -0.01042535],
        [-0.25459535,  0.42726345,  0.29642181, -0.01042535],
        [-0.12525347,  0.3146914 , -0.16509458, -0.32981806],
        [-0.12525347,  0.3146914 , -0.16509458, -0.32981806],
        [-0.23258459,  0.28555156, -0.59729432,  0.11016323],
        [-0.10733113, -0.02913985, -0.43219973,  0.43998129]]),
 array([3.6833216 , 2.23942286, 1.52698177, 1.44445626]),
 array([[-0.63553373, -0.47640777, -0.4613488 , -0.39533505],
        [-0.65996111,  0.25209642,  0.70472713, -0.06525644],
        [ 0.06525644,  0.70472713, -0.25209642, -0.65996111],
        [-0.39533505,  0.4613488 , -0.47640777,  0.63553373]]))
```

```
up, sp, vp = u[:, 0:2], np.diag(s[0:2]), vt[:, 0:2]
```

```
Ap = up @ sp @ vp.T
Ap
```

```
array([[ 1.85910264,  1.61065693, -0.47181162,  0.83189425],
       [ 1.18332367,  0.87547177, -0.43178079,  0.37760539],
       [ 0.71831372,  0.25305371, -0.50656517, -0.05322351],
       [ 0.71831372,  0.25305371, -0.50656517, -0.05322351],
       [ 0.71831372,  0.25305371, -0.50656517, -0.05322351],
       [ 0.18267251,  0.37796321,  0.14657051,  0.30464507],
       [ 0.14013776,  0.86009466,  0.61310485,  0.81215743],
       [ 0.14013776,  0.86009466,  0.61310485,  0.81215743],
       [-0.04253475,  0.48213145,  0.46653434,  0.50751236],
       [-0.04253475,  0.48213145,  0.46653434,  0.50751236],
       [ 0.23980268,  0.7265863 ,  0.3947482 ,  0.63369619],
       [ 0.28233743,  0.24445485, -0.07178614,  0.12618382]])
```

```
Ap = pd.DataFrame(Ap, index=words)
Ap
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **the** | 1.859103 | 1.610657 | -0.471812 | 0.831894 |
| **cow** | 1.183324 | 0.875472 | -0.431781 | 0.377605 |
| **jumped** | 0.718314 | 0.253054 | -0.506565 | -0.053224 |
| **over** | 0.718314 | 0.253054 | -0.506565 | -0.053224 |
| **moon** | 0.718314 | 0.253054 | -0.506565 | -0.053224 |
| **o'leary's** | 0.182673 | 0.377963 | 0.146571 | 0.304645 |
| **kicked** | 0.140138 | 0.860095 | 0.613105 | 0.812157 |
| **lamp** | 0.140138 | 0.860095 | 0.613105 | 0.812157 |
| **started** | -0.042535 | 0.482131 | 0.466534 | 0.507512 |
| **a** | -0.042535 | 0.482131 | 0.466534 | 0.507512 |
| **fire** | 0.239803 | 0.726586 | 0.394748 | 0.633696 |
| **on** | 0.282337 | 0.244455 | -0.071786 | 0.126184 |

```
def sim(w1, w2):
    dot = np.dot(Ap.loc[w1], Ap.loc[w2])
    div = len(Ap.loc[w1]) * len(Ap.loc[w2])
    return dot / div
```

```
sim('the', 'cow')
```

```
0.25799073533786826
```

```
sim('over', 'lamp')
```

```
-0.002218150305832754
```

# Practical 4 – Latent Dirichlet Allocation

[1]
```
import pandas as pd
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
```

[2]
```
documents_list = []
with open("/content/drive/MyDrive/lda_test.txt" ,"r") as fin:
    for line in fin.readlines():
        text = line.strip()
        documents_list.append(text)
```

[3]
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
tokenizer = RegexpTokenizer(r'\w+')

tfidf = TfidfVectorizer(lowercase=True,
                        stop_words='english',
                        ngram_range = (1,1),
                        tokenizer = tokenizer.tokenize)

train_data = tfidf.fit_transform(documents_list)
```

[6]
```
num_components=10

lsa = TruncatedSVD(n_components=num_components, n_iter=100, random_state=42)
lsa.fit_transform(train_data)

Sigma = lsa.singular_values_
V_transpose = lsa.components_.T
```

[7]
```
terms = tfidf.get_feature_names_out()

for index, component in enumerate(lsa.components_):
    zipped = zip(terms, component)
    top_terms_key=sorted(zipped, key = lambda t: t[1], reverse=True)[:5]
    top_terms_list=list(dict(top_terms_key).keys())
    print("Topic "+str(index)+": ",top_terms_list)
```

```
Topic 0:  ['speech', 'text', 'recognition', 'language', 'segmentation']
Topic 1:  ['semantics', 'semantic', 'parsing', 'individual', 'discourse']
Topic 2:  ['semantics', 'individual', 'lexical', 'sentences', 'context']
Topic 3:  ['generation', 'language', 'natural', 'image', 'nlg']
Topic 4:  ['language', 'natural', 'segmentation', 'analysis', 'morphological']
Topic 5:  ['analysis', 'morphological', 'discourse', 'syntactic', 'sentiment']
Topic 6:  ['words', 'named', 'e', 'language', 'word']
Topic 7:  ['segmentation', 'generation', 'morphological', 'topic', 'image']
Topic 8:  ['parsing', 'generation', 'grammar', 'book', 'words']
Topic 9:  ['parsing', 'extraction', 'terminology', 'segmentation', 'relationship']
```

**Practical 5 – POS Tagging**

```
import nltk
import numpy as np
import pandas as pd
import random
```

```
tokens = ["Janet", "will", "back", "the", "bill"]
pos_tags = ["NNP", "MD", "VB", "JJ", "NN", "RB", "DT"]
```

```
A = pd.DataFrame(
    [[0.2767, 0.0006, 0.0031, 0.0453, 0.0449, 0.0510, 0.2026],
    [0.3777, 0.0110, 0.0009, 0.0084, 0.0584, 0.0090, 0.0025],
    [0.0008, 0.0002, 0.7968, 0.0005, 0.0008, 0.1698, 0.0041],
    [0.0322, 0.0005, 0.0050, 0.0837, 0.0615, 0.0514, 0.2231],
    [0.0366, 0.0004, 0.0001, 0.0733, 0.4509, 0.0036, 0.0036],
    [0.0096, 0.0176, 0.0014, 0.0086, 0.1216, 0.0177, 0.0068],
    [0.0068, 0.0102, 0.1011, 0.1012, 0.0120, 0.0728, 0.0479],
    [0.1147, 0.0021, 0.0002, 0.2157, 0.4744, 0.0102, 0.0017]],
    columns = pos_tags,
    index = ["<s>"] + pos_tags )

print(A)
```

```
        NNP      MD      VB      JJ      NN      RB      DT
<s>  0.2767  0.0006  0.0031  0.0453  0.0449  0.0510  0.2026
NNP  0.3777  0.0110  0.0009  0.0084  0.0584  0.0090  0.0025
MD   0.0008  0.0002  0.7968  0.0005  0.0008  0.1698  0.0041
VB   0.0322  0.0005  0.0050  0.0837  0.0615  0.0514  0.2231
JJ   0.0366  0.0004  0.0001  0.0733  0.4509  0.0036  0.0036
NN   0.0096  0.0176  0.0014  0.0086  0.1216  0.0177  0.0068
RB   0.0068  0.0102  0.1011  0.1012  0.0120  0.0728  0.0479
DT   0.1147  0.0021  0.0002  0.2157  0.4744  0.0102  0.0017
```

```
B = pd.DataFrame(
    [[0.000032, 0, 0, 0.000048, 0],
    [0, 0.308431, 0, 0, 0],
    [0, 0.000028, 0.000672, 0, 0.000028],
    [0, 0, 0.000340, 0, 0],
    [0, 0.000200, 0.000223, 0, 0.002337],
    [0, 0, 0.010446, 0, 0],
    [0, 0, 0, 0.506099, 0]],
    columns = tokens,
    index = pos_tags )

print(B)
```

```
        Janet      will      back       the      bill
NNP  0.000032  0.000000  0.000000  0.000048  0.000000
MD   0.000000  0.308431  0.000000  0.000000  0.000000
VB   0.000000  0.000028  0.000672  0.000000  0.000028
JJ   0.000000  0.000000  0.000340  0.000000  0.000000
NN   0.000000  0.000200  0.000223  0.000000  0.002337
RB   0.000000  0.000000  0.010446  0.000000  0.000000
DT   0.000000  0.000000  0.000000  0.506099  0.000000
```

```
# initialize viterbi matrix
viterbi_vals = pd.DataFrame(np.zeros((len(pos_tags), len(tokens))), columns = tokens, index = pos_tags)

associated_tags= {}
for i, token in enumerate(tokens):
  for tag in pos_tags:
    # compute viterbi * a * b and select max
    if i == 0:
      viterbi_vals.at[tag, token] = A.at["<s>", tag] * B.at[tag, token]
    else:
      viterbi_vals.at[tag, token] = max([viterbi_vals.at[p_tag, tokens[i-1]] * A.at[p_tag, tag] * B.at[tag, token] for p_tag in pos_tags])
  associated_tags[token] = viterbi_vals[token].idxmax()

print(viterbi_vals)
```

```
        Janet          will          back           the          bill
NNP  0.000009  0.000000e+00  0.000000e+00  2.486140e-17  0.000000e+00
MD   0.000000  3.004069e-08  0.000000e+00  0.000000e+00  0.000000e+00
VB   0.000000  2.231309e-13  1.608527e-11  0.000000e+00  1.017072e-20
JJ   0.000000  0.000000e+00  5.106917e-15  0.000000e+00  0.000000e+00
NN   0.000000  1.034194e-10  5.359258e-15  0.000000e+00  2.013571e-15
RB   0.000000  0.000000e+00  5.328409e-11  0.000000e+00  0.000000e+00
DT   0.000000  0.000000e+00  0.000000e+00  1.816199e-12  0.000000e+00
```

```
# path
print("\n", associated_tags)
```

```
 {'Janet': 'NNP', 'will': 'MD', 'back': 'RB', 'the': 'DT', 'bill': 'NN'}
```

# Practical 6 – Chunking

```
import nltk
nltk.download()
```

```
# Chinking
grammar = r"""
  NP:
    {<.*>+}          # Chunk everything
    }<VBD|IN>+{      # Chink sequences of VBD and IN
  """
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
       ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]
cp = nltk.RegexpParser(grammar)

print(cp.parse(sentence))
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

```
#  Reading IOB Format and the CoNLL 2000 Corpus
import nltk
nltk.download('conll2000')
from nltk.corpus import conll2000
print(conll2000.chunked_sents('train.txt')[99])
```

```
(S
  (PP Over/IN)
  (NP a/DT cup/NN)
  (PP of/IN)
  (NP coffee/NN)
  ,/,
  (NP Mr./NNP Stone/NNP)
  (VP told/VBD)
  (NP his/PRP$ story/NN)
  ./.)
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data]   Unzipping corpora/conll2000.zip.
```

```
print(conll2000.chunked_sents('train.txt', chunk_types=['NP'])[99])
```

```
(S
  Over/IN
  (NP a/DT cup/NN)
  of/IN
  (NP coffee/NN)
  ,/,
  (NP Mr./NNP Stone/NNP)
  told/VBD
  (NP his/PRP$ story/NN)
  ./.)
```

```
# Simple Evaluation and Baselines

cp = nltk.RegexpParser("")
test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
print(cp.evaluate(test_sents))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  This is separate from the ipykernel package so we can avoid doing imports until
ChunkParse score:
    IOB Accuracy:  43.4%%
    Precision:      0.0%%
    Recall:         0.0%%
    F-Measure:      0.0%%
```

```
grammar = r"NP: {<[CDJNP].*>+}"
cp = nltk.RegexpParser(grammar)
print(cp.evaluate(test_sents))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  This is separate from the ipykernel package so we can avoid doing imports until
ChunkParse score:
    IOB Accuracy:  87.7%%
    Precision:     70.6%%
    Recall:        67.8%%
    F-Measure:     69.2%%
```

## Unigram Chunker

- input--> tree chunk---> list of word--> train unigram tagger
- input--> tagged sent--> IOB tag chunk--> training--> chunk tags--> combine with original sent--> chunk tree--> output

```
[ ] class UnigramChunker(nltk.ChunkParserI):
        def __init__(self, train_sents):
          train_data = [[(t,c) for w,t,c in nltk.chunk.tree2conlltags(sent)]
                        for sent in train_sents]
          self.tagger = nltk.UnigramTagger(train_data)

        def parse(self, sentence):
            pos_tags = [pos for (word,pos) in sentence]
            tagged_pos_tags = self.tagger.tag(pos_tags)
            chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
            conlltags = [(word, pos, chunktag) for ((word,pos),chunktag)
                         in zip(sentence, chunktags)]
            return nltk.chunk.conlltags2tree(conlltags)
```

```
test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
unigram_chunker = UnigramChunker(train_sents)
print(unigram_chunker.evaluate(test_sents))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  after removing the cwd from sys.path.
ChunkParse score:
    IOB Accuracy:  92.9%%
    Precision:     79.9%%
    Recall:        86.8%%
    F-Measure:     83.2%%
```

```
[ ] postags = sorted(set(pos for sent in train_sents for (word,pos) in sent.leaves()))
    print(unigram_chunker.tagger.tag(postags))
```

```
[('#', 'B-NP'), ('$', 'B-NP'), ("'", 'O'), ('(', 'O'), (')', 'O'), (',', 'O'), ('.', 'O'), (':', 'O'), ('CC', 'O'), ('CD', 'I-NP'), ('DT', 'B-NP')
```

```
class BigramChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
      train_data = [[(t,c) for w,t,c in nltk.chunk.tree2conlltags(sent)]
                    for sent in train_sents]
      self.tagger = nltk.BigramTagger(train_data)

    def parse(self, sentence):
        pos_tags = [pos for (word,pos) in sentence]
        tagged_pos_tags = self.tagger.tag(pos_tags)
        chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
        conlltags = [(word, pos, chunktag) for ((word,pos),chunktag)
                     in zip(sentence, chunktags)]
        return nltk.chunk.conlltags2tree(conlltags)
```

```
bigram_chunker = BigramChunker(train_sents)
print(bigram_chunker.evaluate(test_sents))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.

ChunkParse score:
    IOB Accuracy:  93.3%%
    Precision:     82.3%%
    Recall:        86.8%%
    F-Measure:     84.5%%
```

## Training Classifier-Based Chunkers

```
[ ]  class ConsecutiveNPChunkTagger(nltk.TaggerI):

        def __init__(self, train_sents):
            train_set = []
            for tagged_sent in train_sents:
                untagged_sent = nltk.tag.untag(tagged_sent)
                history = []
                for i, (word, tag) in enumerate(tagged_sent):
                    featureset = npchunk_features(untagged_sent, i, history)
                    train_set.append( (featureset, tag) )
                    history.append(tag)
            self.classifier = nltk.MaxentClassifier.train(train_set, trace=0)

        def npchunk_features(sentence, i, history):
          word, pos = sentence[i]
          return {"pos": pos}

        def tag(self, sentence):
            history = []
            for i, word in enumerate(sentence):
                featureset = npchunk_features(sentence, i, history)
                tag = self.classifier.classify(featureset)
                history.append(tag)
            return zip(sentence, history)

    class ConsecutiveNPChunker(nltk.ChunkParserI):
        def __init__(self, train_sents):
            tagged_sents = [[((w,t),c) for (w,t,c) in
                             nltk.chunk.tree2conlltags(sent)]
                            for sent in train_sents]
            self.tagger = ConsecutiveNPChunkTagger(tagged_sents)

        def parse(self, sentence):
            tagged_sents = self.tagger.tag(sentence)
            conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
            return nltk.chunk.conlltags2tree(conlltags)
```

```
chunker = ConsecutiveNPChunker(train_sents)
print(chunker.evaluate(test_sents))
```

```
      Training stopped: keyboard interrupt
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.

ChunkParse score:
    IOB Accuracy:  92.9%%
    Precision:     79.9%%
    Recall:        86.8%%
    F-Measure:     83.2%%
```

## Practical 7- Named Entity Recognition (CRF)

```
import string

# tags = ['company', 'person', 'date', 'amount', 'duration']

text_train = 'Google hired XYZ in 2005, for $100,000 per annum.'
text_train = ''.join([word for word in text_train if word not in string.punctuation]).split(" ")

text_train_tags = {
    'Google'                : 'company',
    'hired'                 : 'vbd',        # verb
    'XYZ'                   : 'person',
    'in'                    : 'cc',         # conjuction
    '2005'                  : 'date',
    'for'                   : 'in',         # preposition
    '100000'                : 'amount',
    'per'                   : 'in',         # preposition
    'annum'                 : 'duration',
}

text_test = 'Apple hired ABC in 1995, for $10 per hour.'
text_test = ''.join([word for word in text_test if word not in string.punctuation]).split(" ")

print("Train data : ", text_train)
print("Test data : ", text_test)
```

```
Train data :  ['Google', 'hired', 'XYZ', 'in', '2005', 'for', '100000', 'per', 'annum']
Test data :  ['Apple', 'hired', 'ABC', 'in', '1995', 'for', '10', 'per', 'hour']
```

```
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.round(e_x / e_x.sum(axis=0), 3)
```

```
# assuming the statement will always start with company
feature_functions = [
    lambda position, label1, label2 : 1 if position == 0 else 0,
    lambda position, label1, label2 : 1 if label1 == 'company' and label2 == 'vbd' else 0,
    lambda position, label1, label2 : 1 if label1 == 'vbd'     and label2 == 'person' else 0,
    lambda position, label1, label2 : 1 if label1 == 'person'  and label2 == 'cc' else 0,
    lambda position, label1, label2 : 1 if label1 == 'cc'      and label2 == 'date' else 0,
    lambda position, label1, label2 : 1 if label1 == 'date'    and label2 == 'in' else 0,
    lambda position, label1, label2 : 1 if label1 == 'in'      and label2 == 'amount' else 0,
    lambda position, label1, label2 : 1 if label1 == 'amount'  and label2 == 'in' else 0,
    lambda position, label1, label2 : 1 if label1 == 'in'      and label2 == 'duration' else 0,
]
```

```
import numpy as np
from sklearn import preprocessing

np.set_printoptions(suppress=True)

print(text_train)


epochs = 30
l_rate = 0.05
weights = np.random.rand(len(feature_functions))
print(weights)

for epoch in range(epochs):
    for token_pos in range(len(text_train)):
        # print(token_pos, train_tokens[token_pos])
        if text_train[token_pos] in text_train_tags:
            # print(token_pos, train_tokens[token_pos])
            weights += l_rate * np.multiply(weights,
                                            [f(token_pos,
                                               text_train_tags[text_train[max(0, token_pos-1)]],
                                               text_train_tags[text_train[token_pos]])
                                            for f in feature_functions])
    print(f"Epoch {epoch} : ", softmax(weights))

weights = preprocessing.normalize([weights])
```

```
['Google', 'hired', 'XYZ', 'in', '2005', 'for', '100000', 'per', 'annum']
[0.9679286  0.0365424  0.83346896 0.79120149 0.38337629 0.23688606
 0.67048689 0.98796373 0.273651  ]
Epoch 0 :  [0.158 0.06  0.137 0.132 0.086 0.073 0.116 0.162 0.076]
Epoch 1 :  [0.161 0.058 0.139 0.132 0.084 0.072 0.116 0.164 0.075]
Epoch 2 :  [0.163 0.056 0.14  0.133 0.083 0.07  0.116 0.167 0.073]
Epoch 3 :  [0.166 0.053 0.141 0.134 0.081 0.068 0.115 0.17  0.071]
Epoch 4 :  [0.168 0.051 0.142 0.134 0.08  0.066 0.115 0.173 0.069]
Epoch 5 :  [0.171 0.049 0.143 0.135 0.078 0.064 0.115 0.176 0.068]
Epoch 6 :  [0.174 0.047 0.144 0.136 0.077 0.062 0.115 0.179 0.066]
Epoch 7 :  [0.177 0.045 0.145 0.137 0.075 0.06  0.114 0.183 0.064]
Epoch 8 :  [0.181 0.043 0.147 0.137 0.073 0.058 0.114 0.186 0.062]
Epoch 9 :  [0.184 0.04  0.148 0.138 0.071 0.056 0.113 0.19  0.059]
Epoch 10 :  [0.188 0.038 0.149 0.139 0.069 0.054 0.113 0.194 0.057]
Epoch 11 :  [0.191 0.036 0.15  0.139 0.067 0.051 0.112 0.198 0.055]
Epoch 12 :  [0.195 0.034 0.151 0.14  0.065 0.049 0.111 0.202 0.053]
Epoch 13 :  [0.199 0.031 0.152 0.14  0.063 0.047 0.11  0.207 0.05 ]
Epoch 14 :  [0.203 0.029 0.154 0.141 0.06  0.044 0.109 0.212 0.048]
Epoch 15 :  [0.207 0.027 0.155 0.141 0.058 0.042 0.108 0.217 0.046]
Epoch 16 :  [0.212 0.025 0.155 0.141 0.055 0.04  0.107 0.222 0.043]
Epoch 17 :  [0.216 0.023 0.156 0.141 0.053 0.037 0.106 0.227 0.041]
Epoch 18 :  [0.221 0.021 0.157 0.141 0.05  0.035 0.104 0.232 0.038]
Epoch 19 :  [0.226 0.019 0.158 0.141 0.048 0.032 0.102 0.238 0.036]
Epoch 20 :  [0.23  0.017 0.158 0.141 0.045 0.03  0.101 0.244 0.033]
Epoch 21 :  [0.236 0.015 0.159 0.14  0.043 0.028 0.099 0.25  0.031]
Epoch 22 :  [0.241 0.014 0.159 0.14  0.04  0.025 0.097 0.256 0.029]
Epoch 23 :  [0.246 0.012 0.159 0.139 0.037 0.023 0.094 0.262 0.026]
Epoch 24 :  [0.251 0.011 0.159 0.138 0.035 0.021 0.092 0.269 0.024]
Epoch 25 :  [0.257 0.009 0.159 0.137 0.032 0.019 0.089 0.276 0.022]
Epoch 26 :  [0.262 0.008 0.159 0.136 0.03  0.017 0.086 0.283 0.02 ]
Epoch 27 :  [0.268 0.007 0.158 0.134 0.027 0.015 0.083 0.29  0.018]
Epoch 28 :  [0.273 0.006 0.157 0.132 0.025 0.013 0.08  0.297 0.016]
Epoch 29 :  [0.279 0.005 0.156 0.13  0.022 0.012 0.077 0.304 0.014]
```

```
# predict
tags = list(set(text_train_tags.values()))
print(tags)

prev = 'company'
pred_dict = {}

for token_pos in range(1, len(text_test)):
    probs = []
    for tag in tags:
        feature_out = [f(token_pos, prev, tag) for f in feature_functions]
        mult = np.multiply(weights, feature_out)
        probs.append(sum(mult[0]))
    probs = softmax(probs)
    prev = tags[list(probs).index(max(probs))]
    pred_dict[text_test[token_pos]] = [round(i, 3) for i in probs]

for i in pred_dict.items():
    print(i)
```

```
['person', 'cc', 'company', 'vbd', 'date', 'amount', 'in', 'duration']
('hired', [0.125, 0.125, 0.125, 0.127, 0.125, 0.125, 0.125, 0.125])
('ABC', [0.178, 0.117, 0.117, 0.117, 0.117, 0.117, 0.117, 0.117])
('in', [0.118, 0.175, 0.118, 0.118, 0.118, 0.118, 0.118, 0.118])
('1995', [0.122, 0.122, 0.122, 0.122, 0.148, 0.122, 0.122, 0.122])
('for', [0.123, 0.123, 0.123, 0.123, 0.123, 0.123, 0.139, 0.123])
('10', [0.117, 0.117, 0.117, 0.117, 0.117, 0.164, 0.117, 0.134])
('per', [0.116, 0.116, 0.116, 0.116, 0.116, 0.116, 0.19, 0.116])
('hour', [0.117, 0.117, 0.117, 0.117, 0.117, 0.164, 0.117, 0.134])
```

# Practical 8 - BERT

```
[1] !pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
```

```
[4] from transformers import BertTokenizer, BertForQuestionAnswering
    import torch

    tokenizer = BertTokenizer.from_pretrained("deepset/bert-base-cased-squad2")
    model = BertForQuestionAnswering.from_pretrained("deepset/bert-base-cased-squad2")

    question, text = "When is Independence day in India", "Independence day is on 15th August"

    inputs = tokenizer(question, text, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs)

    answer_start_index = outputs.start_logits.argmax()
    answer_end_index = outputs.end_logits.argmax()

    predict_answer_tokens = inputs.input_ids[0, answer_start_index : answer_end_index + 1]
```

```
[5] print("Question:",question,"\nANS:",tokenizer.decode(predict_answer_tokens))
```

```
Question: When is Independence day in India
ANS: on 15th August
```

## Practical 9 –Sentiment Analysis

### Sentiment analysis

```
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb
import numpy as np
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = 20000)
```

```
x_train = sequence.pad_sequences(x_train, maxlen=50)
x_test = sequence.pad_sequences(x_test, maxlen=50)
```

```
model = Sequential()
model.add(Embedding(20000, 128))
model.add(LSTM(128, dropout = 0.2, recurrent_dropout = 0.2))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, None, 128)         2560000

 lstm_1 (LSTM)                (None, 128)               131584

 dense_1 (Dense)              (None, 1)                 129

=================================================================
Total params: 2,691,713
Trainable params: 2,691,713
Non-trainable params: 0
```

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
model.fit(x_train, y_train, batch_size = 32, epochs = 5, validation_data = (x_test, y_test))
```

```
Epoch 1/5
782/782 [==============================] - 151s 189ms/step - loss: 0.4652 - accuracy: 0.7738 - val_loss: 0.4066 - val_accuracy: 0.8134
Epoch 2/5
782/782 [==============================] - 141s 180ms/step - loss: 0.2926 - accuracy: 0.8788 - val_loss: 0.4081 - val_accuracy: 0.8152
Epoch 3/5
782/782 [==============================] - 140s 180ms/step - loss: 0.1922 - accuracy: 0.9252 - val_loss: 0.5261 - val_accuracy: 0.8100
Epoch 4/5
782/782 [==============================] - 141s 180ms/step - loss: 0.1203 - accuracy: 0.9555 - val_loss: 0.6555 - val_accuracy: 0.7935
Epoch 5/5
782/782 [==============================] - 141s 180ms/step - loss: 0.0789 - accuracy: 0.9724 - val_loss: 0.6745 - val_accuracy: 0.7915
<keras.callbacks.History at 0x7f3bc84e0290>
```

```
op = model.predict(x_test)
for i in range(20):
    print(op[i], y_test[i])
```

```
[0.1568169] 0
[0.9925916] 1
[0.5112637] 1
[0.58092415] 0
[0.9999993] 1
[0.5088515] 1
[0.86745715] 1
[0.00489676] 0
[0.00434095] 0
[0.9338963] 1
[0.9413802] 1
[0.00208059] 0
[5.8518137e-05] 0
[0.02026719] 0
[0.95651007] 1
[0.00045055] 0
[0.88902986] 1
[0.00185642] 0
[1.8383807e-05] 0
[0.00723889] 0
```

```
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (key, value) in index.items()])

for i in range(10):
    print(" ".join([reverse_index.get(i - 3, "#") for i in x_train[i]]))
    print(y_train[i], model.predict(x_train[i].reshape(1,50)))
    print("----------------------------------------")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was som
1 [[0.9782326]]
----------------------------------------
boobs and # taking away bodies and the gym still doesn't close for # all joking aside this is a truly bad film whose only charm is to look back on the disaster that was the 80's and have a good old laugh a
0 [[0.00075382]]
----------------------------------------
must have looked like a great idea on paper but on film it looks like no one in the film has a clue what is going on crap acting crap costumes i can't get across how # this is to watch save yourself an hou
0 [[0.00736728]]
----------------------------------------
man to see a film that is true to scotland this one is probably unique if you maybe # on it deeply enough you might even re evaluate the power of storytelling and the age old question of whether there are
1 [[0.97924864]]
----------------------------------------
the # and watched it burn and that felt better than anything else i've ever done it took american psycho army of darkness and kill bill just to get over that crap i hate you sandler for actually going thro
0 [[0.00662354]]
----------------------------------------
# # # # # # # begins better than it ends funny that the russian submarine crew # all other actors it's like those scenes where documentary shots br br spoiler part the message # was contrary to the whole
0 [[0.00584954]]
----------------------------------------
which has been toned down a bit in its harsh # i liked the look of the film and how shots were set up and i thought it didn't rely too much on # of head shots like most other films of the 80s and 90s do ve
1 [[0.9891043]]
----------------------------------------
anyone br br the hamiltons commits the cardinal sin of being both dull boring from which it never recovers add to that an ultra thin story no gore a rubbish ending character's who you don't give a toss abo
0 [[0.00317359]]
----------------------------------------
void was also a great doc about mountain climbing and showing the intensity in an engaging way but this film is much more of a human story i just saw it today but i will go and say that this is one of the
1 [[0.999948]]
----------------------------------------
dash of campy ridiculousness then pop a bowl of popcorn invite some friends over and have some fun br br i agree with other comments that the sound is very bad dialog is next to impossible to follow much o
0 [[0.00988078]]
```

```
review1 = "dash of campy ridiculousness then pop a bowl of popcorn invite some friends over and have some fun br br i agree with other comments that the sound
review2 = "void was also a great doc about mountain climbing and showing the intensity in an engaging way but this film is much more of a human story i just sa
encoded1 = np.array([[index[token] for token in review1.split(" ")]])
encoded2 = np.array([[index[token] for token in review2.split(" ")]])

# higher means negative
print(model.predict(encoded1))
print(model.predict(encoded2))
```

```
[[0.98153853]]
[[0.10830122]]
```